Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias en Ciencias de la Computación

Recuperación de información utilizando códigos de Hadamard y el grafo de semi-espacios proximales

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de

Maestro en Ciencias

Presenta:

Bryan Rodrigo Quiroz Palominos

Ensenada, Baja California, México

2021

Tesis defendida por

Bryan Rodrigo Quiroz Palominos

y aprobada por el siguiente Comité

Dr. Edgar Leonel Chávez González

Director de tesis

Dr. Irvin Hussein López Nava

Dr. Daniel Sauceda Carvajal

Dr. Pedro Gilberto López Mariscal

Coordinador del Posgrado en Ciencias de la Computación

Dr. Pedro Negrete Regagnon

Director de Estudios de Posgrado

Resumen de la tesis que presenta Bryan Rodrigo Quiroz Palominos como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Recuperación de información utilizando códigos de Hadamard y el grafo de semi-espacios proximales

Resumen aprobado por:

Dr. Edgar Leonel Chávez González
Director de tesis

Las redes neuronales convolucionales (CNN, del inglés *Convolutional Neural Networks*) han obtenido buenos resultados en tareas de clasificación de imágenes. Las últimas capas de los modelos de CNN permiten responder y crear sus propios filtros con patrones más complejos en la entrada, como texturas, formas o variaciones de características procesadas en capas anteriores. Los sistemas de recuperación de imágenes tienen como objetivo navegar en colecciones mediante consultas. Debido a la cantidad de atributos y elementos que componen una imagen, es complicado extraer características de la imagen de forma manual. Estos sistemas extraen de modelos de CNN preentrenados un vector representativo de cada imagen de la colección. Este vector, por lo general, es extraído de la penúltima capa y se le conoce como las características profundas (del inglés *Deep features*). Los vectores con características profundas tienen una alta dimensión, por ejemplo, el vector de características profundas extraído de un modelo ResNet tiene 2,048 dimensiones. Además, los valores del vector son números reales. Esto provoca que los sistemas consuman más memoria principal (RAM), ya que los datos se indexan en índices que funcionan en memoria principal. También requieren más ciclos de cómputo para realizar cálculos. En este trabajo de tesis, se propone la construcción de un encaje binario que permite mejorar la representación de imágenes con características profundas. Se integraron los códigos de Hadamard en la arquitectura de ResNet para obtener un vector binario en codificación de Hadamard. Su eficacia se compara con la representación con características profundas mediante experimentos en tareas de recuperación utilizando los conjuntos de Places365 e ImageNet. Además, se propone el grafo de semi-espacios proximales (HSP, del inglés *Half Space Proximal*) para recuperar imágenes, el cual se compara con el algoritmo de k vecinos más cercanos mediante el cálculo de la pureza. Los resultados obtenidos muestran que la representación de Hadamard es en promedio mejor que la representación con características profundas en tareas de recuperación utilizando el grafo HSP. También, se muestra que, para calcular la distancia entre vectores, se reduce la huella de memoria y se necesitan menos ciclos de cómputo. Por último, se realizó una implementación de un prototipo de un sistema de recuperación de imágenes, en donde se incluyeron distintas herramientas para realizar consultas.

Palabras clave: sistemas de recuperación de imágenes, encaje de imágenes, códigos de Hadamard, grafo de semi-planos proximales

Abstract of the thesis presented by Bryan Rodrigo Quiroz Palominos as a partial requirement to obtain the Master of Science degree in Computer Science.

Information retieval using Hadamard codes and the half-space proximal graph

Abstract approved by:

Dr. Edgar Leonel Chávez González
Thesis Director

Convolutional Neural Networks (CNNs) have shown good performance on image classification. The last layers of the architecture of the CNN models allow them to create their filters with more complex patterns in the input, such as textures, shapes, or variations of features processed in previous layers. Image retrieval systems aim to navigate collections with queries. Due to the number of attributes and objects that make up an image, it is challenging to extract image features manually. These systems extract from pre-trained CNN models a representative vector of each image in the collection. This vector is usually extracted from the penultimate layer and is known as the deep features. Deep features vectors have a high dimension. In ResNet, the Deep features vector extracted has 2,048 dimensions. In addition, the values of the vector are real numbers. This causes systems to consume more main memory, since data is usually indexed in indexes that run in the main memory. Also, more computation cycles are required to perform calculations. In this thesis work, we propose the development of an image binary embedding that improves the deep features image representation. Hadamard codes were integrated into the ResNet architecture to obtain a binary vector in Hadamard encoding. Its efficiency is compared with the Deep features representation with experiments in retrieval tasks using Places365 and ImageNet datasets. In addition, the Half Space Proximal graph is proposed to retrieve images, which is compared with the k nearest neighbor algorithm by calculating the purity. The results obtained show that the Hadamard representation is, on average, better than the deep features representation in retrieval tasks using the HSP graph. Also, it shows that the memory footprint is reduced for computing the distance between vectors, and fewer computation cycles are needed. Finally, a prototype of an image retrieval was implemented, including different tools to make queries.

Keywords: image retrieval systems, image embedding, Hadamard codes, half-space proximal graph

Dedicatoria

A mi familia, especialmente a mis papás y hermanas que siempre me apoyan incondicionalmente, me alientan a esforzarme cada día más y por ser parte crucial en cada etapa de mi vida.

Agradecimientos

A mis papás, por apoyarme en todas mis decisiones, por su cariño, por alentarme a crecer en el ámbito personal y profesional. Gracias, son un ejemplo para mí y son mi fortaleza.

A mis hermanas, Erika y Valeria, por enseñarme como superar los obstáculos de la vida y siempre ser un soporte crucial en mi vida.

A mis abuelos, tíos y primos, por apoyarme y aconsejarme.

A mi director de tesis, el Dr. Edgar Leonel Chávez González, por darme la oportunidad de contribuir a su trabajo de investigación, su constante orientación, su disponibilidad, su apoyo incondicional y su confianza en mí.

A los miembros de mi comité de tesis, el Dr. Irvin Hussein López Nava y el Dr. Daniel Sauceda Carvajal, por su tiempo, sus correcciones de tesis, sus comentarios, su paciencia, su retroalimentación y por enriquecer mi trabajo de tesis.

A mis amigos, especialmente Yayo, Alan, Sandez, Martel, Tuyín y Roy, por brindarme su apoyo, comprensión y ayudarme a relajarme en esos momentos difíciles.

A mis compañeros de posgrado, por su amistad y apoyo.

A todas aquellas personas que estuvieron presentes durante esta etapa.

Al Centro de Investigación Científica y de Educación Superior de Ensenada por la oportunidad de realizar mis estudios de posgrado.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar mis estudios de maestría. No. de becario: 995814.

Tabla de contenido

Página

Tabla de contenido (continuación)

Tabla de contenido (continuación)

Lista de figuras

Figura Página

Lista de figuras (continuación)

Lista de tablas

Tabla Página

Capítulo 1. Introducción

Las colecciones multimedia, también conocidas como bases de datos multimedia, se han convertido en un recurso de datos fundamental para múltiples áreas tanto científicas como industriales. Dado que las personas han recibido herramientas para la producción, el almacenamiento y el intercambio de contenido multimedia, la creación y el consumo de contenido multimedia se ha incrementado exponencialmente en los últimos años. Estas han ayudado a gestionar el vasto crecimiento de información multimedia (Alemu *et al.*, 2009) almacenando diferentes tipos de información como texto, imágenes, audios o videos. Por tal motivo, uno de los mayores retos es su escala, es decir, tanto el número de elementos como el espacio utilizado.

Este abrupto crecimiento conlleva una extensa gama de problemas, por ejemplo, dificultades en el acceso a la información, escalabilidad, datos no estructurados (es decir, carecen de indexación semántica o temática), entre otros. Lo anterior ha provocado la proliferación en la investigación de sistemas de navegación multimedia, que contienen decenas o hasta miles de millones de imágenes (Khan, 2020). Estos sistemas permiten realizar consultas para recuperar objetos de una colección multimedia. Una consulta es una petición precisa de un objeto que se desea recuperar.

En la Figura 1 se muestra un ejemplo del funcionamiento generalizado de un sistema de navegación multimedia. Este es similar a la arquitectura de red de cliente-servidor (Sulyman, 2014) en donde el servidor contiene recursos y el cliente, por medio de consultas, solicita estos recursos. En los sistemas de navegación multimedia el usuario sería el cliente que realiza peticiones de recursos mediante consultas y el servidor contiene la lógica para acceder a la información almacenada en la colección multimedia cada vez que tenga que responder a una petición de un recurso. Esta arquitectura permite centralizar de manera lógica todos los recursos para un mejor funcionamiento, eficiencia, y accesibilidad. En la práctica el sistema puede estar distribuido en varias computadoras físicas y para el usuario aparece como un solo recurso lógico. Un ejemplo paradigmático es la máquina de búsqueda Google. Los usuarios finales del sistema esperan trabajar con colecciones multimedia de gran escala, incluso con recursos informáticos limitados o con los que tienen a su disposición.

Los sistemas de navegación multimedia se integran fundamentalmente de cuatro

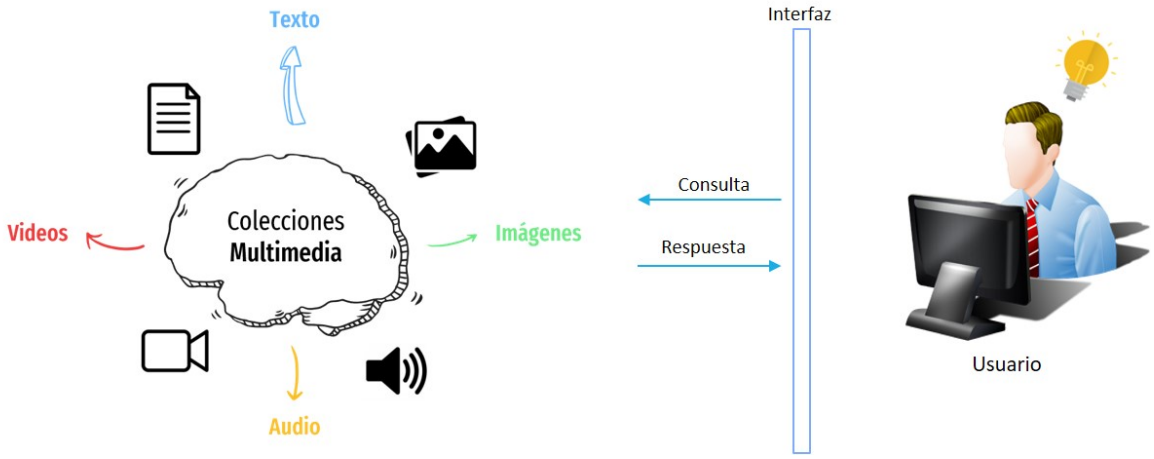


Figura 1. Funcionamiento general de un sistema de navegación multimedia.

componentes: (1) Sistema de indexación: Compuesto por métodos y procedimientos de indexación y búsqueda; (2) Colección de objetos: Imágenes, videos o cualquier otro tipo de objeto multimedia; (3) Herramientas para realizar consultas: Conjunto definido de consultas que se ingresan en el sistema y permiten navegar en la colección de objetos, pueden ser con o sin la participación del usuario; y (4) Criterios de evaluación: Medidas utilizadas para evaluar la precisión, recuperación y relevancia de los objetos recuperados en respuesta a la consulta (Tanner, 2002).

Un sistema de navegación multimedia debe cumplir con ciertos requerimientos mínimos (Véase la Figura 2). Estos pueden dividirse en dos grandes grupos, aquellos orientados al desempeño del sistema y los encargados de la relevancia de información presentada al usuario. En el primer grupo tenemos el requerimiento de **Rapidez**: El usuario debe recibir los resultados sugeridos en segundos, **Interactividad**: Permite una interacción a modo de diálogo entre el sistema y el usuario, y por último, tenemos la **Escalabilidad**: El usuario debe poder interactuar incluso con colecciones de gran escala. El segundo grupo está compuesto por **Relevancia**: Los elementos sugeridos deben ser relevantes a la consulta del usuario, **Comprensibilidad**: Permite al usuario comprender si el sistema es adecuado y cómo puede ser utilizado para tareas específicas, y por último, la **Adaptabilidad**: Es un mecanismo que permite responder a los cambios internos y externos a través del tiempo.

La interacción del usuario es una tarea que representa un papel fundamental en el problema de navegación (Huang *et al.*, 2008). Este problema se encuentra suma-

Figura 2. Requerimientos de un sistema de navegación multimedia.

mente relacionado con el problema de clasificación. Podemos definir el problema de clasificación como el proceso de predecir una clase (comúnmente llamadas etiquetas o categorías) de un conjunto de datos utilizando un modelo predictivo de clasificación en el que se aproxima una función de mapeo (f) de las variables de entrada (X) a las variables de salida discreta (y). Sin embargo, la diferencia con el problema de navegación radica en que no es una función la que realiza el mapeo, sino que el usuario es el encargado de determinar si un elemento es relevante o no; siendo la principal razón de la importancia del usuario en este proceso.

Los sistemas de navegación multimedia requieren implementar algoritmos que permitan a los usuarios recuperar objetos de manera rápida y eficiente de una colección. El aprendizaje interactivo es uno de los primeros algoritmos propuestos por la comunidad científica para recuperar imágenes y videos basados en contenido. Recientemente estos algoritmos han tenido un resurgimiento dado que son capaces de satisfacer múltiples necesidades de navegación; permitiendo desde exploración hasta la búsqueda de un elemento en particular. Estos algoritmos utilizan la retroalimentación (o *feedback*, en inglés) del usuario para seguir mostrando objetos relevantes. Mediante la retroalimentación es posible adaptarse a la intención de búsqueda y conocimiento del usuario; generando un lazo entre sistema y usuario.

La dificultad para encontrar un objeto dado (sea imagen, video, texto o audio) es muy alta. Pensemos, por ejemplo, en buscar un concepto en un diccionario, la persona usualmente busca una palabra para saber su significado en un diccionario. Esto sería equivalente a buscar los metadatos de una imagen. Hay ocasiones en las que

el usuario sabe el concepto y quisiera conocer la palabra que determina el concepto; por ejemplo, supongamos que el usuario quiere saber si existe una palabra que describa lo siguiente: «*Realizar alguna acción para que el sentido del discurso sea más fácil de comprender, con alguna aclaración o con una explicación.*» el usuario debería encontrar que la palabra *desambiguar* es el concepto que requiere. Esto es lo que se conoce como una búsqueda inversa, en la que de una cantidad N de resultados, se va acotando hasta llegar al resultado.

Para imágenes y videos sucede algo similar, en ocasiones se requiere buscar una imagen con detalles específicos y lo que generalmente se realiza es: (1) asociar la imagen a un concepto (si se requiere una imagen con sol, arena y mar, podría asociarse a una playa) y (2) escribir una descripción detallada de la imagen. Las dos opciones anteriores pueden caer en ambigüedades, la primera opción podría asociarse a un concepto incorrecto y convertirse en una búsqueda errónea; terminando en el peor de los casos en volver a iniciar la búsqueda. La segunda opción puede ser complicado para una persona describir lo que busca.

Una buena alternativa para realizar búsquedas y qué funciona mejor es mostrar elementos semejantes a lo que se busca, esto se conoce como la búsqueda dado un ejemplo. De esta manera se presentan elementos y el usuario interactúa decidiendo las que son relevantes para él. Este tipo de interacciones permite utilizar las capacidades del usuario para hacer síntesis de la semejanza percibida.

En los diferentes trabajos de investigación se ha estudiado el uso de algoritmos de aprendizaje interactivo para la búsqueda de imágenes y videos (Ragnarsdóttir *et al.* (2019); Kratochvíl *et al.* (2020)) obteniendo buenos resultados. A partir de la información de estos trabajos se han identificado beneficios al utilizar alguna variante que utilice la retroalimentación del usuario.

El sorprendente crecimiento en aprendizaje profundo (del inglés *deep learning*), específicamente en tareas de clasificación de imágenes, ha permitido obtener las llamadas características profundas (del inglés *deep features*). Estas son un subproducto del proceso de clasificación de imágenes y han ayudado significativamente en tareas de recuperación multimedia que anteriormente representaban un reto.

Aunado a esto el uso de modelos previamente entrenados de aprendizaje profundo

permiten ahorrar tiempo, tanto en código como en recursos informáticos. Además, en lugar de entrenar una red neuronal profunda desde cero para una tarea específica permite: (1) Tomar una red entrenada en un dominio diferente para una tarea de origen diferente; y (2) Adaptarlo para un dominio y tarea objetivo. Obteniendo variaciones para un mismo dominio, tarea diferente o diferente dominio, misma tarea.

Las redes neuronales convolucionales (CNN por sus siglas en inglés, *Convolutional Neural Networks*) se han utilizado para la clasificación y predicción de imágenes de manera eficaz. Inicialmente eran utilizadas como otro clasificador, peleando contra otros clasificadores populares de la época como las máquinas de soporte vectorial (del inglés *Support Vector Machines*). Después tuvieron un salto cualitativo y cuantitativo con las arquitecturas profundas. La investigación en visión computacional se enfocaba en técnicas de extracción de características. Actualmente, una red neuronal diseñada para clasificar imágenes es entrenada con un conjunto de imágenes y dentro de la arquitectura aprende filtros (o características) como detectores de bordes y contornos de imágenes de capas anteriores. Las capas más profundas pueden responder y crear sus propios filtros de características para patrones más complejos en la entrada, como texturas, formas o variaciones de características procesadas anteriormente; estas características son conocidas como características profundas (del inglés *Deep Features*) y pueden ser utilizadas para realizar clasificación de imágenes.

1.1. Antecedentes

El problema de navegación en colecciones multimedia consiste en encontrar maneras de indexación de datos, procesamiento de objetos (imágenes, videos, audio o texto), algoritmos de navegación y herramientas de interfaz gráfica para realizar consultas. Sin embargo, la tarea de navegación resulta ser compleja y no solo depende de contar con un buen *backend* (sistema de indexación, colección de objetos y criterios de evaluación) sino que el usuario juega un rol importante en el que debe ser capaz de lograr encontrar el objeto deseado de una manera rápida, precisa y sencilla.

La competencia de *Video Browser Showdown* (VBS por sus siglas en inglés) surge a raíz de la necesidad de intentar proporcionar tareas altamente desafiantes sobre una colección de videos (Schoeffmann, 2019). Cabe destacar que en este trabajo se

realizará navegación y recuperación sobre una colección de imágenes, pero resulta importante resaltar a esta competencia por su indudable empuje al crecimiento en el área de investigación en sistemas de navegación multimedia. Además, la mayoría de los sistemas presentados en dicha competencia y del estado del arte han sido adaptados para navegar en colecciones de vídeos e imágenes (específicamente en el área de *Lifelogging* (Gurrin *et al.*, 2021)).

Dentro de la competencia se definen entornos de evaluación justos y esquemas de puntuación de desempeño para conocer la efectividad de los sistemas participantes durante las evaluaciones públicas. Los equipos participantes intentan resolver tantas tareas como sea posible en modo experto (usuarios que tienen conocimiento del sistema) y sesiones de novatos, cada tarea es presentada a todos los equipos en la misma habitación al mismo tiempo con el mismo límite de tiempo.

La competencia evalúa dos tipos diferentes de tareas de búsqueda. La búsqueda de elementos conocidos (KIS, del inglés *Known-Item Search*) es una situación en la que los usuarios conocen el elemento que desean buscar. La simulación se realiza presentando la escena objetivo real y se solicita a los equipos participantes que encuentren la escena en un tiempo máximo de 5 minutos (entre menos tiempo requerido, mejor). También se presentan otro tipo de situaciones en la que se presenta una descripción textual de la escena objetivo con un límite de tiempo de 8 minutos. Asimismo, esta última situación permite evaluar cuando los usuarios no tienen claro el objetivo que buscan pero cuentan con características que les permiten llegar al objetivo. En igual forma la segunda tarea denominada Búsqueda de video ad-hoc (del inglés *Ad-hoc Video Search*) los equipos intentan encontrar la mayor cantidad de escenas relacionadas a una clase o tópico (por ejemplo, un niño en la playa) en un tiempo límite de 5 minutos.

Ambas tareas se realizan para usuarios expertos y novatos. Generalmente los usuarios expertos son los programadores del sistema y los novatos son personas que conocen el funcionamiento del sistema, pero antes de iniciar con la evaluación se permite darle un breve recorrido por parte de los programadores. La sesión de novatos permite evaluar aspectos de la usabilidad del sistema.

Con las redes neuronales se tiene la ventaja de poder extraer características de

las capas más profundas. La mayoría de los sistemas presentados en VBS utilizan la representación de objetos multimedia con características profundas. Sin embargo, una de sus mayores desventajas es el tamaño de la representación que proporcionan las diferentes arquitecturas de redes neuronales convolucionales. Por ejemplo, ResNet50 proporciona un vector de dimensionalidad 512, VGG16 proporciona un vector de dimensionalidad 4,096 y ResNet101 proporciona un vector de dimensionalidad de 2,048. Lo anterior puede representar una desventaja al momento de construir un sistema de navegación multimedia, ya que indexar una cantidad considerable de datos con una representación de gran tamaño podría requerir más memoria RAM de la que se cuenta disponible en ese momento. Esto afecta directamente al indexamiento de los datos y al mantenimiento del índice.

1.2. Planteamiento del problema

Una imagen puede estar compuesta por objetos, los cuales expresan tamaño, color, forma, textura, ubicación, dominio, relación espacial entre objetos, entre otros aspectos. La cantidad de atributos que puede contener una imagen, vuelve compleja la tarea de realizar representaciones de imágenes de forma manual. Sin embargo, las redes neuronales profundas mediante la técnica de extracción de las características profundas han facilitado este trabajo. Por lo general, estas características se extraen de la penúltima capa de un modelo de aprendizaje profundo y se obtiene un vector de N dimensiones. Este vector contiene los propios filtros generados por la red neuronal que representan patrones más complejos de la imagen, como texturas, formas o variaciones de características procesadas anteriormente.

Las representaciones de imágenes con características profundas tienen la propiedad de poder obtener imágenes semejantes con el calculo de la distancia euclidiana entre vectores. Si la distancia es pequeña, quiere decir que la imagen es semejante. Si la distancia es grande, entonces la imagen no es semejante. Esta propiedad se utiliza en sistemas de recuperación de imágenes. Sin embargo, realizar el proceso para obtener imágenes semejantes puede convertirse en una tarea lenta y que consume mucha memoria RAM, dependiendo la cantidad de imágenes con la que se trabaje. Esto porque se requiere comparar una consulta contra todas las imágenes de la colección. El uso de índices métricos resulta ser una buena alternativa para acelerar la búsqueda

de imágenes semejantes, sin embargo, se ejecutan en memoria RAM, lo que provoca aumentar aún más el consumo de memoria RAM.

1.3. Propuesta de solución

En esta tesis, se abordará el problema de navegar en colecciones multimedia, específicamente en el dominio de imágenes. Se utilizarán modelos de aprendizaje profundo preentrenados para extraer las características profundas. Estos modelos preentrenados abren la posibilidad a realizar etiquetado de imágenes sin la necesidad de reentrenamiento. Sin embargo, las características profundas obtenidas de modelos preentrenados son de alta dimensión, lo cual provoca que el proceso para encontrar elementos semejantes sea lento. Por lo cual, se propone mejorar la representación de las características profundas construyendo un encaje de imágenes que permita reducir la huella de memoria y los ciclos de cómputo al recuperar imágenes semejantes en un sistema de recuperación de imágenes. Además, el encaje debe mantener la propiedad de semejanza que permite recuperar objetos semejantes calculando la distancia entre los objetos de la colección.

1.4. Justificación de la investigación

Las colecciones multimedia concentran información a través de una variedad de archivos de texto, audio, imágenes, o videos. Esta información resulta ser muy provechosa para distintas aplicaciones, abriendo la posibilidad a desarrollar aplicaciones que sean útiles para diseñadores gráficos, curadores de arte, colecciones de museos, entre otros. En el caso específico de imágenes, se pueden desarrollar aplicaciones para colecciones de fotografías para periódicos, televisoras, revistas y medios informativos en general.

El tamaño de las colecciones multimedia ha aumentado exponencialmente, ocasionando que la infraestructura de los sistemas de recuperación necesite mayor cantidad de recursos de cómputo. Por tal motivo, es importante encontrar alternativas para representar imágenes que consuman menos memoria y que requieran menos ciclos de cómputo al recuperar imágenes semejantes.

1.5. Hipótesis

- La representación de imágenes con códigos de Hadamard al utilizar el grafo de semi-espacios proximales tiene un mejor desempeño en tareas de recuperación en los conjuntos de datos de Places365 e ImageNet que usar la representación con características profundas.
- Al utilizar la representación de imágenes con códigos de Hadamard y la representación con características profundas en los conjuntos de datos de Places365 e ImageNet, los vecinos relevantes recuperados con el grafo de semi-espacios proximales tienen una mayor pureza que los recuperados con el algoritmo de k vecinos más cercanos.

1.6. Objetivos

1.6.1. Objetivo general

Mejorar la representación de imágenes con características profundas construyendo un encaje binario que permita comparar imágenes de tal manera que, imágenes semejantes tengan distancia pequeña e imágenes distintas tengan distancia grande.

1.6.2. Objetivos específicos

- Comparar la eficacia de la representación de Hadamard y la representación de características profundas utilizando diferentes conjuntos de datos y algoritmos.
- Determinar la eficacia de la representación de Hadamard en tareas de recuperación en colecciones multimedia de imágenes utilizando el grafo de semi-espacios proximales.
- Desarrollar un prototipo de un sistema de recuperación de imágenes.

1.7. Resultados esperados

Se espera desarrollar un prototipo de un sistema de recuperación de imágenes que permita hacer experimentos de recuperación y visualización de imágenes. Al lograr

una representación binaria de imágenes se podría reducir la huella de memoria producida por la representación de imágenes con características profundas. Además,

Capítulo 2. Marco teórico

Un sistema de recuperación de imágenes tiene como objetivo encontrar las imágenes semejantes a una imagen dada como consulta entre las imágenes de una base de datos. Sus principales funcionalidades son la navegación, búsqueda y recuperación de imágenes (Datta *et al.*, 2008).

La tarea de recuperación de imágenes es un proceso complejo que requiere técnicas de diferentes campos, como el reconocimiento de patrones, recuperación de información y aprendizaje automático. El método tradicional y más utilizado para la recuperación de imágenes es asignar metadatos, tales como un título, palabras clave o descripciones a las imágenes de tal manera que pueda realizarse la búsqueda por texto con un alto nivel de precisión. Sin embargo, asignar metadatos de forma manual a cientos o miles de millones de imágenes es un proceso que consume mucho tiempo, es laborioso y caro. Para atacar este problema existe una gran cantidad de investigación en la anotación automática de imágenes (del inglés *Automatic image annotation*) que se encarga de investigar cómo asignar automáticamente metadatos en forma de subtítulos o palabras clave a una imagen (Hervé y Boujemaa (2007); Zhang *et al.* (2012)).

La ventaja de la búsqueda por texto mediante consultas que incluyan términos, conceptos o palabras clave es que permite a los usuarios realizar consultas de una manera más natural, ya que es la manera más común de búsqueda en muchos sistemas que se utilizan diariamente. Sin embargo, el principal problema de este tipo de búsqueda, radica en: describir todos los atributos de una imagen, ya que una imagen puede contener una gran variedad de atributos, como tamaño, color, figuras, texturas, localización, posición, dominio, entre otros.

2.1. Recuperación de imágenes por contenido

La recuperación de imágenes por contenido (CBIR, del inglés *Content based image retrieval*) es un problema que se enfoca en buscar imágenes semánticamente semejantes o coincidentes en una base de datos de gran escala mediante el análisis del contenido visual de una imagen dada por el usuario (Lew *et al.*, 2006).

La principal diferencia entre los sistemas de recuperación basados en texto y los basados en el contenido, es que en este último, la interacción humana resulta indispensable. Los humanos tienden a utilizar características de alto nivel, como conceptos, palabras o descripciones para interpretar las imágenes y medir su similitud. Sin embargo, las características que utilizan los sistemas de recuperación de imágenes por contenido son de bajo nivel, ya que se enfocan en el color, textura, figuras y relaciones espaciales de una imagen. Desde la proliferación de las redes neuronales convolucionales, la extracción de características de bajo nivel se ha convertido en una tarea más sencilla.

Diferentes técnicas de consulta se han implementado en sistemas que utilizan el modelo de recuperación de imágenes por contenido. Entre las más implementadas tenemos la consulta dado un ejemplo y aprendizaje de interactivo.

2.1.1. Consulta dado un ejemplo

La consulta dado un ejemplo es una técnica que consiste en proporcionar al sistema una imagen de ejemplo que será utilizada para realizar una búsqueda en la base de datos. Los algoritmos utilizados pueden variar entre aplicación, sin embargo, comparten la característica que deben encontrar imágenes con características de bajo nivel semejantes al ejemplo proporcionado (Zhu y Wu, 2014). La Figura 3 presenta la visión que tiene la consulta dado un ejemplo.

El sistema debe proveer al usuario las herramientas necesarias para proporcionar un ejemplo. Algunos enfoques permiten a los usuarios proporcionar una imagen de su computadora y utilizarla como ejemplo. También es posible presentar un conjunto aleatorio de imágenes y que el usuario seleccione una imagen que satisfaga su necesidad de búsqueda. Otro enfoque es permitir al usuario dibujar un bosquejo utilizando un pequeño lienzo, en donde se dibujen figuras de la imagen que se busca.

Esta técnica busca eliminar para el usuario la dificultad que existe al tener que describir una imagen con palabras, conceptos o descripciones.

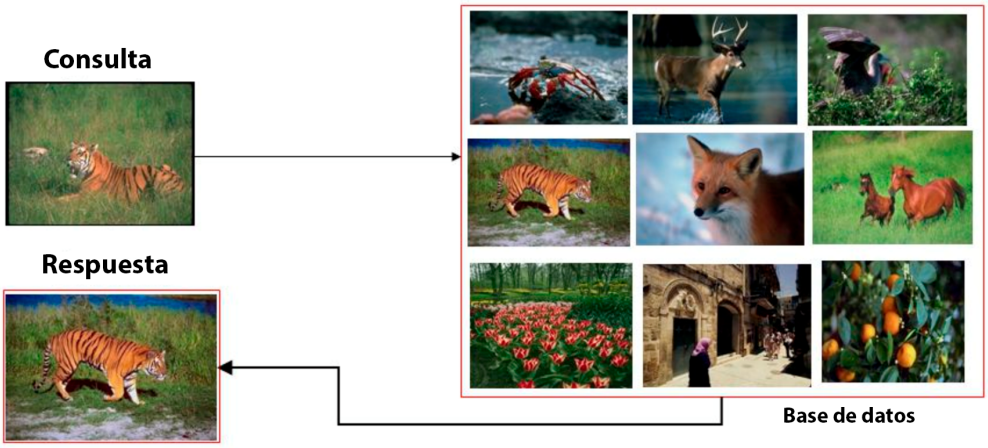


Figura 3. Ejemplo de una consulta dado un ejemplo. Podemos visualizar que existe una relación entre la consulta y la respuesta, ya que la consulta contiene un tigre acostado y la respuesta contiene un tigre en movimiento. Esto expresa el objetivo que busca cumplir está técnica, encontrar imágenes semejantes al ejemplo dado.

2.1.2. Aprendizaje interactivo

El aprendizaje interactivo es una técnica de recuperación que ha recibido gran atención por los sistemas de recuperación por contenido. Esto debido a que la interacción humana es una parte esencial en el proceso de recuperación, lo cual permite su uso en tareas de navegación (Huang *et al.*, 2008). Para recuperar objetos se presenta al usuario un conjunto inicial de imágenes, las cuales deben etiquetarse o seleccionarse como «relevantes» o «no relevantes». La retroalimentación otorgada por el usuario mediante la selección de imágenes se utiliza como estrategia de búsqueda para ajustar los resultados; a esto se le conoce como ronda de retroalimentación. Cada ronda de retroalimentación pretende acercar al usuario a su objetivo. El proceso termina hasta que el usuario encuentre un elemento que satisfaga sus necesidades o decida terminar el proceso.

Generalmente, los sistemas que implementan este enfoque son diseñados de manera que la interfaz contenga dos columnas, las cuales permiten seleccionar imágenes como «relevantes» o «no relevantes».

2.2. Redes Neuronales Convolucionales

Las redes neuronales convolucionales, también llamadas CNN o ConvNets (por su nombre en inglés, *Convolutional Neural Networks*), es una clase de red neuronal artificial (ANN por su nombre en inglés, *Artificial Neural Network*), que comúnmente se utiliza para analizar imágenes visualmente (Valueva *et al.*, 2020). Tienen aplicaciones en reconocimiento de imágenes y videos, sistemas de reconocimiento, clasificación de imágenes, segmentación de imágenes, análisis de imágenes médicas, procesamiento del lenguaje natural, interfaces cerebro-computadora, y series temporales financieras.

Las CNNs están compuestas por capas o niveles (Goodfellow *et al.*, 2016). Las capas más cercanas a la entrada se enfocan en el reconocimiento de estructuras simples. Mientras que las capas siguientes usan las estructuras simples para reconocer o detectar estructuras más complejas. Las CNNs se forman usando tres tipos de capas: (1) Capas convolucionales: Requiere el uso de máscaras y filtrado, (2) Capas de *pooling*: consiste en reducir las representaciones obtenidas de manera que se hagan más pequeñas y sean más manejables computacionalmente, reduciendo el número de parámetros, y por último, (3) Capas totalmente conectadas (del inglés *Fully connected layers*): Se aplican al final y se pierde la información espacial. La Figura 4 muestra la estructura básica de una red neuronal convolucional.

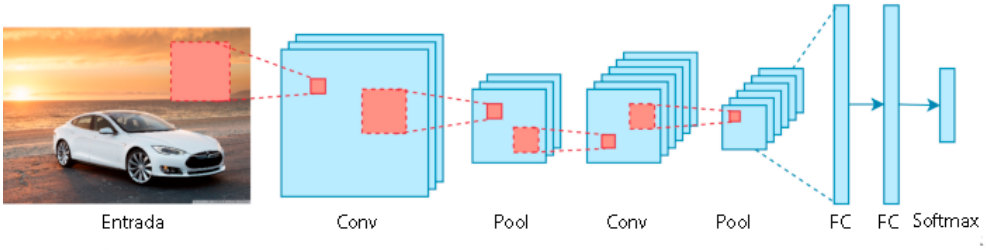


Figura 4. Arquitectura básica de una red neuronal convolucional.

La principal ventaja de las CNNs es que pueden detectar un objeto a pesar de su posición espacial en la imagen. Esto provoca que una CNN sea capaz de detectar un objeto cuando se encuentra en una posición diferente o el objeto se encuentra girado en diferentes ángulos.

El nombre de «red neuronal convolucional» indica que la red emplea una operación matemática llamada «convolución». La convolución es un tipo de operación lineal. Las

redes neuronales convolucionales son simplemente redes neuronales que utilizan la convolución en lugar de la multiplicación matricial en al menos una de sus capas.

La convolución consiste en tomar un grupo de píxeles de la imagen de entrada e ir realizando un producto escalar con un kernel. Este kernel recorrerá todas las neuronas de entrada y se obtendrá una nueva matriz. En caso de que la imagen sea a color, se tendrán 3 kernels del mismo tamaño para cada uno de los canales. A medida que se utilicen más capas con convoluciones, los mapas de características serán capaces de reconocer formas más complejas.

El preprocesamiento requerido en una ConvNet es mucho menor en comparación con otros algoritmos de clasificación. Mientras que en los métodos tradicionales los filtros se diseñan a mano. Con suficiente entrenamiento, las ConvNet tienen la capacidad de aprender estos filtros/características. Este algoritmo de aprendizaje profundo puede tomar una imagen como entrada, asignar importancia (pesos y sesgos aprendibles) a varios aspectos/objetos en la imagen y ser capaz de diferenciar unos de otros.

Las arquitecturas básicas de CNNs se pueden agrupar en dos grandes grupos, las que entregan una salida para toda la imagen (comúnmente utilizadas para clasificación) y las que contienen salida totalmente conectada (las que entregan una salida por píxel, utilizadas en segmentación de imágenes).

2.2.1. Arquitecturas

El rendimiento de las CNNs depende en gran medida de sus modelos. Para lograr el mejor rendimiento, las arquitecturas de CNNs del estado del arte fueron construidas manualmente por expertos que tienen gran conocimiento en el desarrollo de modelos. El proyecto ImageNet (Deng *et al.*, 2009) es una gran base de datos de imágenes diseñada para su uso en la investigación de reconocimiento de objetos. Desde el año 2010, el proyecto ImageNet cuenta con una competencia titulada *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*, en donde los programas de software compiten para clasificar y detectar correctamente objetos y escenas. ImageNet se ha convertido en un marco de referencia en el área de clasificación, detección de objetos y escenas. Esto ha generado que surjan nuevas arquitecturas de CNNs. La mayoría de las arquitecturas con un buen rendimiento en el marco de referencia de ImageNet se han

convertido en el estado del arte del área.

2.2.1.1. LeNet

LeNet-5 es una red neuronal convolucional pionera de siete niveles realizada por LeCun *et al.* (1989) en 1998; aunque la idea nace de la publicación de 1989, no fue tras varios años de investigación e iteraciones exitosas, que el trabajo recibió el nombre de LeNet-5. Fue utilizada para clasificar dígitos escritos a mano en imágenes de entrada de 32x32 píxeles en escala de grises. La resolución de las imágenes fue tan pequeña, ya que para una mayor resolución se requería una mayor cantidad de capas convolucionales, sin embargo, con los recursos informáticos de la época, era imposible aumentar el número de capas.

2.2.1.2. AlexNet

AlexNet (Krizhevsky *et al.*, 2012) fue la primera red neuronal convolucional profunda. Publicada en el año 2012, logró superar significativamente a todos los competidores anteriores de la competencia de ImageNet.

AlexNet obtuvo una precisión del 84.7 % comparada con el segundo lugar que obtuvo una precisión del 73.8 %. La red es muy similar a LeNet, pero más profunda, con más filtros por capa y con capas convolucionales apiladas. La estructura de AlexNet consiste en 5 capas convolucionales y 3 capas totalmente conectadas. Además, de utilizar ReLu como función de activación. La red cuenta con 62 millones de parámetros entrenables.

La entrada de la red es un lote de imágenes RGB de tamaño 227x227x3 y su salida es un vector de probabilidad de 1000x1, que corresponde cada una de las 1000 clases de ImageNet.

AlexNet se entrenó durante 6 días simultáneamente en dos GPUs Nvidia Geforce GTX 580, razón por la cual esta red se encuentra dividida en dos *pipelines*.

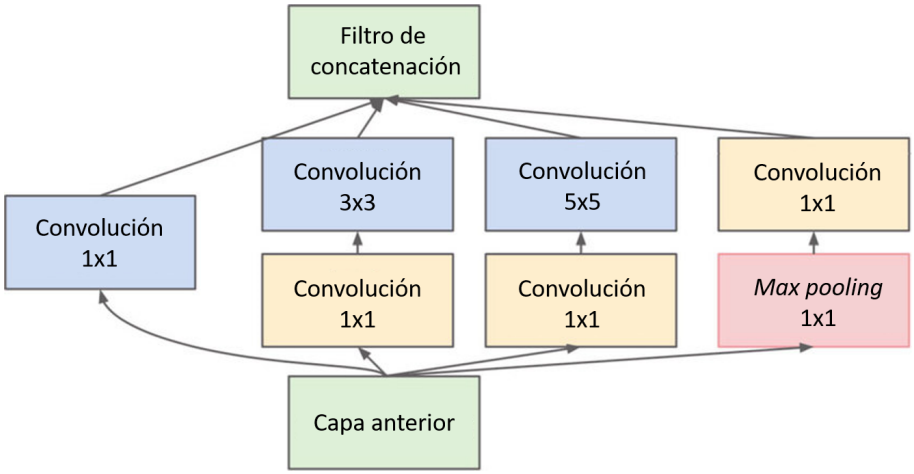


Figura 5. Módulo *inception* de la CNN GoogLeNet. Los bloques convolucionales 1x1 mostrados en amarillo son utilizados para reducción de profundidad. El resultado de las cuatro operaciones en paralelo se concatenan en profundidad para formar el bloque de concatenación mostrado en verde (Szegedy *et al.*, 2015).

2.2.1.3. GoogLeNet/Inception

El ganador de la competencia de ImageNet del año 2014 fue GoogLeNet (también conocida como *Inception v1*) de Google (Szegedy *et al.*, 2015). La red se inspiró en LeNet, pero implementó un elemento novedoso que se denomina módulo de origen (en inglés *inception module*), de ahí viene el nombre de la red.

La arquitectura de la red GoogLeNet consiste en múltiples módulos *inception*. Cada módulo consiste en cuatro operaciones en paralelo que contienen una capa convolucional 1x1, una capa convolucional 3x3, una capa convolucional 5x5 y max pooling. La Figura 5 presenta de manera gráfica un módulo *inception*.

El módulo *inception* tiene como objetivo principal reducir drásticamente el número de parámetros. La arquitectura de GoogLeNet consiste de veintidós capas convolucionales que permite reducir el número de parámetros de sesenta millones que contenía AlexNet a solamente cuatro millones.

2.2.1.4. VGGNet

VGGNet (Simonyan y Zisserman, 2014) fue el subcampeón de la competencia de ImageNet del año 2014, en donde el ganador fue GoogLeNet. Existen múltiples varian-

tes de esta red, por ejemplo VGG16 y VGG19 que sólo difieren en el número total de capas de la red. La estructura es muy similar a la red AlexNet con la diferencia que utiliza capas convolucionales 3x3, pero con muchos filtros.

Actualmente es la opción preferida para extraer características de imágenes. Sin embargo, consta de 138 millones de parámetros, lo cual puede ser un reto de manejar.

2.2.1.5. ResNet

La red residual (ResNet, del inglés *Residual Network*) es la red ganadora de la competencia ImageNet 2015. Esta red introduce un concepto que se inspira en el hecho biológico de que algunas neuronas se conectan con otras neuronas que no se encuentran contiguas, saltando capas intermedias (He *et al.*, 2016). Las redes neuronales residuales logran lo anterior utilizando atajos o «conexiones de salto» para moverse sobre varias capas; a esto se le conoce como Bloque residual (Véase la Figura 6). El salto entre capas elimina las complicaciones de la red, volviéndolas más simples y usando muy pocas capas durante la etapa de entrenamiento. Además, acelera el aprendizaje hasta en diez veces, minimizando el efecto de desvanecimiento de gradiente.

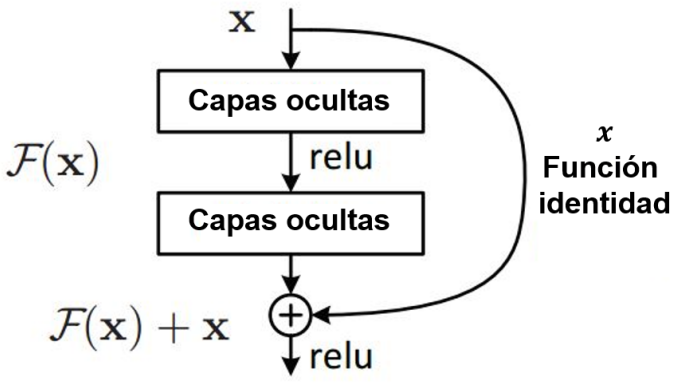


Figura 6. La arquitectura ResNet utiliza conexiones de salto para resolver el problema de desvanecimiento de gradiente. La imagen muestra un bloque residual básico de ResNet que se repite en toda la red (He *et al.*, 2016).

Existen múltiples versiones de ResNet con la diferencia principal que varia el número de capas. Las más comunes son ResNet50 y ResNet101, pero también existen versiones como ResNet19 y ResNet152. El mayor impacto de ResNet se debe a que su

paradigma implementado permitió por primera vez entrenar redes muy profundas (de más de 100 capas).

La Tabla 1 presenta una comparación entre las CNN con mayor precisión en la base de datos ImageNet. De los datos de cada CNN se pueden hacer las siguientes comparaciones:

Tabla 1. Comparación de CNNs con su top-5 de precisión en la base de datos de ImageNet.

Red	Año	Top 5 precisión	% de Error	# de parámetros	FLOP
AlexNet	2012	84.70 %	11.7	62 millones	1.5 B
Inception	2014	93.30 %	6.7	6.4 millones	2B
VGGNet	2014	92.30 %	7.3	138 millones	19.6 B
ResNet-152	2015	95.51 %	3.6	60.3 millones	11 B

FLOP: Operaciones de punto flotante por segundo (del inglés *Floating Point Operations per Second*)

- AlexNet y ResNet-152 cuentan con un número de parámetros semejantes; aún cuando existe un 10 % de precisión de diferencia. Sin embargo, el entrenamiento es más demandante en ResNet.
- VGGNet es la red con más parámetros y que requiere más tiempo de entrenamiento, pero no es la red con mejor precisión.
- AlexNet es la red que menos tiempo de entrenamiento requiere, pero es la red con peor precisión.
- *Inception* es la red con la menor cantidad de parámetros. Requiere casi 10 veces menos tiempo de entrenamiento que VGGNet y es la segunda que mejor precisión tiene, por debajo de ResNet.

Una de las desventajas de las arquitecturas presentadas es que requieren de un gran poder de cómputo y miles de imágenes por categoría para lograr el rendimiento publicado, limitando su uso en otras aplicaciones.

2.2.2. Características profundas

Las redes neuronales convolucionales han mostrado buenos resultados en tareas de clasificación de imágenes. Sin embargo, el entrenamiento de una red profunda desde

cero no es una opción viable cuando se quiere resolver un problema de clasificación con un pequeño número de ejemplos de entrenamiento etiquetados. Para resolver este problema es común el uso de la transferencia de aprendizaje (*Transfer Learning, en inglés*). Este método se utiliza en el aprendizaje automático para almacenar el conocimiento adquirido al resolver un problema en particular y utilizar ese mismo conocimiento para resolver una tarea diferente.

Las características profundas (del inglés *Deep Features*) (Pan y Yang, 2010) es la obtención de vectores de *N* dimensiones de la penúltima capa de la red neuronal. Para ello, es necesario eliminar la etapa de clasificación (capa *softmax*) de la red, que corresponde a la última capa de la red. Estas características profundas contienen filtros con características de patrones más complicados en una imagen, como texturas, formas o variaciones de características procesadas anteriormente. Las representaciones de imágenes aprendidas por estás redes son potentes y genéricas. Estas representaciones genéricas se han utilizado para resolver numerosos problemas de reconocimiento visual (Razavian *et al.* (2014); Donahue *et al.* (2014)). Dado el buen rendimiento de las características profundas, se han convertido en una opción para resolver problemas de visión por computadora (Azizpour *et al.*, 2015). Además, es una técnica muy utilizada para recuperar objetos en sistemas de navegación multimedia (Kratochvíl *et al.* (2020); Ragnarsdóttir *et al.* (2019)). La salida de la penúltima capa pueden utilizarse no sólo para distinguir las caras de objetos no faciales, sino para crear nuevos clasificadores, teniendo la propiedad de que cuando se calcula la distancia euclidiana entre vectores es posible obtener imágenes semejantes.

Las arquitecturas presentadas en la sección 2.2.1 se pueden utilizar para etiquetar colecciones de imágenes extrayendo las características profundas. El vector resultante es una representación de cada imagen de la colección. Esta técnica es muy utilizada en el ámbito de sistemas de recuperación y permite recuperar elementos semejantes.

2.3. Detección de objetos

La detección de objetos es el término que se utiliza para describir las técnicas de visión por computadora que se encargan de identificar y localizar objetos en imágenes digitales. Esta técnica se utiliza para contar los objetos en una escena y determinar su

ubicación. Los objetos identificados son delimitados por recuadros (del inglés *bounding boxes*) que enmarcan el área donde se encuentra el objeto detectado en una escena (Jiao *et al.*, 2019).

Muy a menudo se confunde el reconocimiento de imágenes con la detección de objetos. El reconocimiento de imágenes es la tarea de asignar una etiqueta o clase a una imagen. Por ejemplo, si una imagen contiene un «carro» la etiqueta deseada sería «carro». Si la imagen contiene «dos carros», la etiqueta seguirá siendo «carro». Mientras que en la detección de objetos, se busca dibujar un recuadro delimitador en cada objeto identificado con su respectiva etiqueta. La Figura 7 muestra la diferencia entre el reconocimiento de imágenes y la detección de objetos.

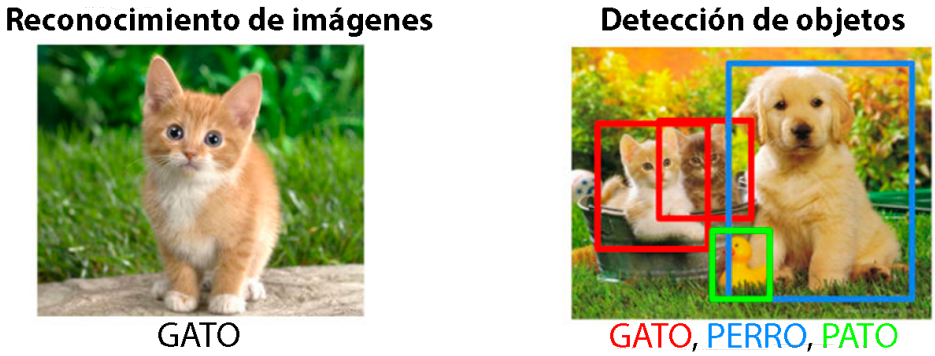


Figura 7. Ejemplo de la diferencia entre el reconocimiento de imágenes y la detección de objetos.

Los modelos de detección de objetos basados en aprendizaje profundo suelen contar con dos partes. Un codificador (del inglés *encoder*) que recibe una imagen como entrada y la procesa por una serie de capas y bloques para extraer características usadas para localizar y etiquetar objetos. La salida del codificador se procesa por un decodificador, que predice los recuadros delimitadores y las etiquetas de cada objeto identificado.

La idea básica de un algoritmo para resolver el problema de detección y localización de múltiples objetos es el proceso de ventana deslizante (del inglés *sliding window algorithm*) (Lampert *et al.*, 2008), que consiste en recortar una imagen en múltiples partes y clasificarlas utilizando una CNN. Sin embargo, esta solución es computacionalmente costosa y no existe garantía alguna de que las ventanas coincidan perfectamente con un objeto. Los siguientes algoritmos han sido desarrollados con la finalidad de encontrar y delimitar objetos rápidamente.

2.3.1. R-CNN

Para evitar el problema de seleccionar una gran cantidad de regiones, (Girshick *et al.*, 2014) propusieron un método llamado *regiones de características con redes neuronales convolucionales*, mejor conocido como R-CNN por sus siglas en inglés. Este método utiliza búsqueda selectiva para extraer dos mil regiones de una imagen, las cuales se les denomina como *regiones propuestas*. Por lo tanto, en vez de clasificar una gran número de regiones, ahora solamente se trabaja con dos mil regiones. El algoritmo de búsqueda selectiva para generar las dos mil regiones propuestas se describe a continuación:

1. Se genera una segmentación inicial, en la cual se generan muchas regiones candidatas.
2. Se utiliza un algoritmo voraz para combinar recursivamente regiones similares en otras más grandes.
3. Por último, las regiones generadas se utilizan como propuestas finales de regiones candidatas.

Las dos mil regiones propuestas se deforman en un cuadrado y se introducen en una red neuronal convolucional que produce un vector de características de salida de 4,096 dimensiones. Las características extraídas se introducen en una máquina de soporte vectorial (SVM) para clasificar un objeto dentro de una región candidata. También el algoritmo predice cuatro valores que representan el recuadro delimitador del objeto identificado.

Los problemas que tiene este método de detección de objetos es que aunque se reduzcan las regiones a dos mil, el entrenamiento de la red sigue requiriendo una gran cantidad de tiempo. Tampoco puede ser implementada en tiempo real porque necesita aproximadamente cuarenta y siete segundos para procesar una imagen, y por último, el algoritmo de búsqueda selectiva podría generar malos candidatos de regiones propuestas que conduciría a una detección de objetos ineficiente.

El autor de R-CNN resolvió con la implementación de Fast R-CNN (Girshick, 2015) algunos de los problemas mencionados. Con esta nueva implementación se logró dis-

minuir el tiempo de entrenamiento significativamente, ya que no hay que alimentar cada vez la red neuronal convolucional con dos mil regiones propuestas.

2.3.2. Faster R-CNN

Faster R-CNN (Ren *et al.*, 2017), conocida también como *regiones de características con redes neuronales convolucionales más rápidas*, a diferencia de las implementaciones de R-CNN o Fast R-CNN, es que elimina el uso del algoritmo de búsqueda selectiva y le permite a la red neuronal conocer las regiones propuestas.

El proceso de este método empieza proporcionando una imagen como entrada a una red convolucional, la cual proporciona un mapa de características. Después se emplea una red de propuestas (RPN) para predecir las propuestas de la región. Las regiones propuestas predichas se modelan mediante una capa de agrupación de rol que se utiliza para clasificar la imagen dentro de la región propuesta y así predecir los valores de los recuadros delimitadores. Este proceso se puede observar en la Figura 8.

2.3.3. YOLO

Los algoritmos de detección de objetos anteriores usan regiones para localizar un objeto dentro de una imagen, por lo cual la CNN no ve la imagen original. Esto puede provocar que algún objeto no se detecte.

El algoritmo «Solo miras una vez» (YOLO, del inglés *You Only Look Once*) es uno de los algoritmos de detección de objetos más potentes del estado del arte. Se llama así porque a diferencia de los algoritmos de detección de objetos como R-CNN o su actualización Faster R-CNN, sólo necesita que la imagen (o el video) pase una vez por la red. YOLO divide la imagen en regiones, prediciendo cuadros de identificación y probabilidades para cada región (Redmon *et al.*, 2016).

El funcionamiento de YOLO se muestra en la Figura 9 y se puede describir en los siguientes pasos:

1. La imagen de entrada se divide en una cuadrícula SxS.

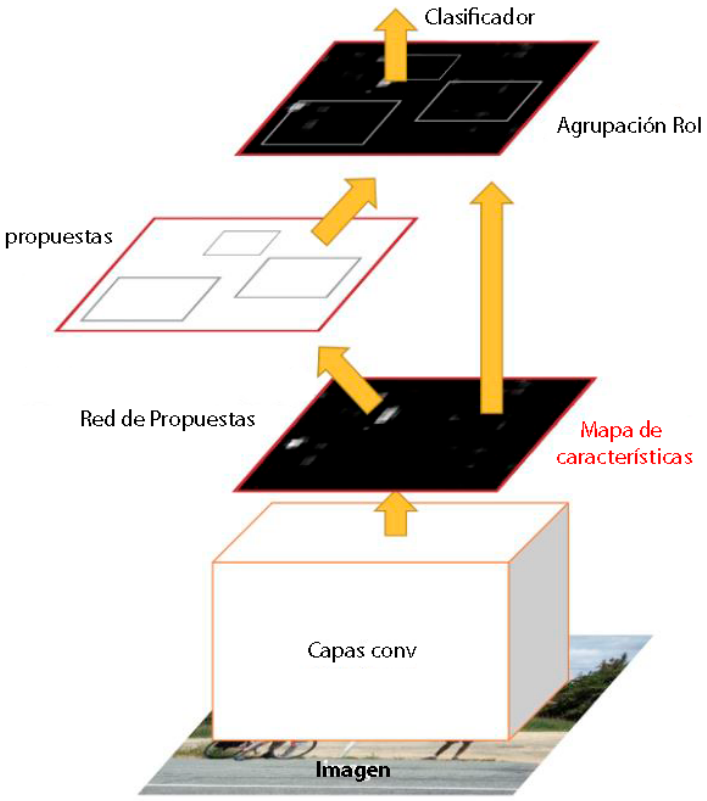


Figura 8. Se muestra el funcionamiento del método Faster R-CNN en una imagen (Ren *et al.*, 2017).

2. En cada cuadrícula se toman m recuadros delimitadores.
3. Para cada recuadro delimitador, se genera una probabilidad de clase y valores para el recuadro delimitador.
4. Los recuadros delimitadores que tengan una probabilidad de clase superior a un valor de umbral especificado, se utilizan para localizar el objeto dentro de la imagen.

YOLO funciona a 45 cuadros por segundo y existen implementaciones que funcionan a mayores velocidades. El principal problema de este algoritmo es la dificultad para detectar objetos pequeños.

La primera versión de YOLO fue publicada en el año 2015. Actualmente existen diferentes versiones de YOLO. En 2016, se presentó YOLO9000: mejor, más rápido, más fuerte (Redmon y Farhadi, 2017) cuyo modelo está entrenado en 9000 clases. YOLOv3 (Redmon y Farhadi, 2018) es la versión más reciente. También pueden encontrarse

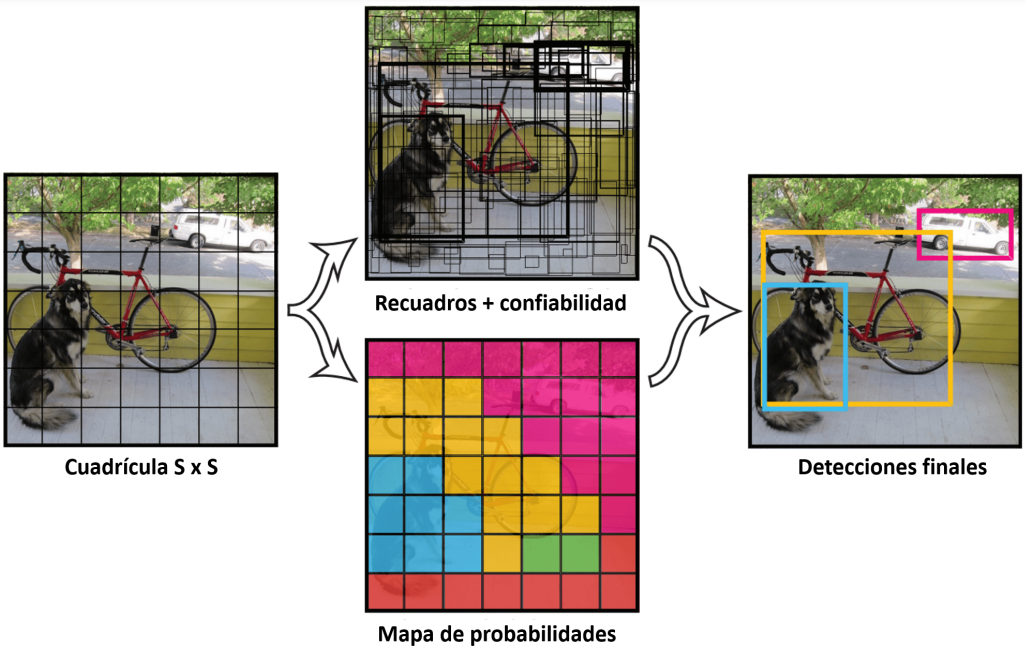


Figura 9. Se muestra una imagen de entrada que divide en una cuadrícula S x S. Se generan los recuadros de limitación y se predice una clase entre las de mayor probabilidad (Redmon *et al.*, 2016).

otras versiones, como YOLOv4 y YOLOv5; aunque estas versiones no fueron desarrolladas por el mismo autor que realizó las primeras tres versiones.

2.4. Segmentación semántica de imágenes

La segmentación de imágenes es un problema fundamental en visión por computadora. Su funcionamiento es segmentar la entrada visual para procesarla en tareas como la clasificación de imágenes y detección de objetos. Los métodos actuales de segmentación de imágenes pueden clasificarse en tres categorías: segmentación semántica, segmentación de instancias y segmentación panóptica (Garcia-Garcia *et al.*, 2017). La Figura 10 muestra ejemplos para cada tipo de segmentación.

Para explicar las diferencias entre las tres categorías de segmentación de imágenes tomaré como referencia la imagen (a) de la Figura 10. La imagen contiene personas, automóviles, edificios, calles, señales de tránsito, etc. Si sólo se necesita agrupar los objetos que pertenecen a una misma categoría, por ejemplo, distinguir todos los automóviles que contiene la imagen de los edificios, se estaría realizando la tarea de segmentación semántica. Dentro de cada categoría, por ejemplo, automóviles, si se

quiere distinguir a cada automóvil individualmente, esa tarea se refiere a la segmentación de instancias. En cambio, si se desea una división tanto por categorías como por instancias, se estaría realizando una tarea de segmentación panóptica.

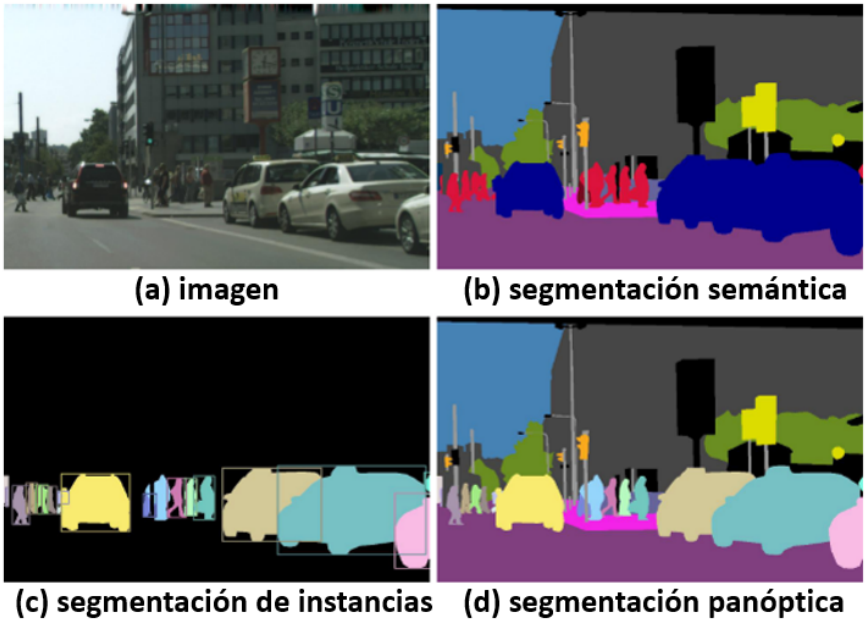


Figura 10. Para una (a) imagen, se muestra el ground truth de: (b) segmentación semántica (una etiqueta de clase por pixel), (c) segmentación de instancias (etiqueta y máscara por cada objeto) y (d) segmentación panóptica (etiqueta de clase e instancia por pixel) (Kirillov *et al.*, 2019).

Hablando técnicamente, la segmentación semántica significa etiquetar cada píxel de una imagen con una clase, la segmentación de instancias se refiere a etiquetar y segmentar cada objeto por una determinada clase y por último, la segmentación panóptica sería etiquetar cada píxel de la imagen con una etiqueta de clase e instancia.

Existen dos convenciones básicas que se siguen en una tarea de segmentación de imágenes:

1. **Objeto**: es cualquier entidad contable, tales como persona, automóvil, flor, entre otros.
2. **Cosa**: es una región amorfa incontable, de textura idéntica, tales como: arena, pared, entre otros.

La segmentación semántica tiene un gran impacto en aplicaciones del mundo real,

tales como automóviles autónomos, imágenes satélites, tiendas de ropa y exploraciones médicas.

2.4.1. Arquitecturas

La arquitectura básica en segmentación de imágenes consiste en un codificador (en inglés *encoder*) y un decodificador (en inglés *decoder*). El codificador extrae características de la imagen mediante filtros. Mientras que el decodificador se encarga de generar la salida final, que suele ser una máscara de segmentación que contiene el contorno del objeto. La mayoría de las arquitecturas se basan en ella, o una variante de ella.

2.4.1.1. Red totalmente convolucional

La red totalmente convolucional (FCN, del inglés *Fully Convolutional Networks*) es un algoritmo muy popular para realizar segmentación semántica (Long *et al.*, 2015). Este modelo utiliza varios bloques de capas de convolución y *max pool* para comprimir una imagen 32 veces su tamaño original. Después se realiza una predicción de clase a este nivel de granularidad. Por último. utilizan capas de muestreo y deconvolución para redimensionar la imagen a sus dimensiones originales. En la Figura 11 se muestra un ejemplo de la red totalmente convolucional.

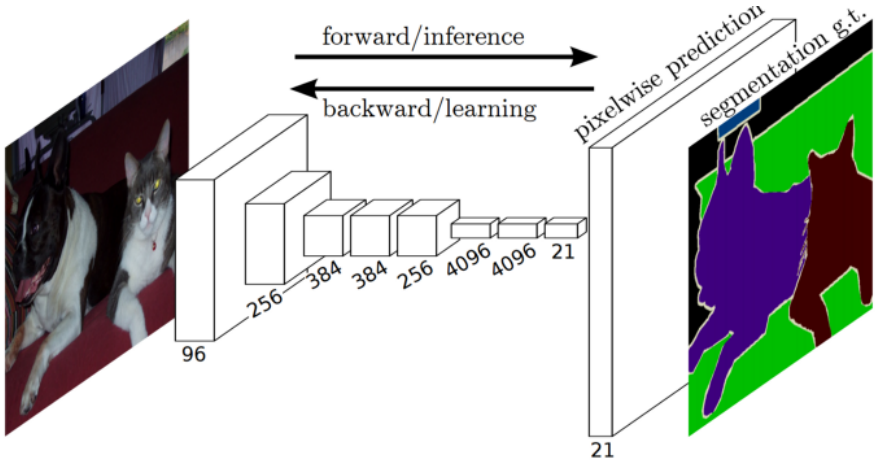


Figura 11. Ejemplo de la red totalmente convolucional (Long *et al.*, 2015).

Estos modelos no cuentan con capas totalmente conectadas. El objetivo del muestreo descendente es capturar la información semántica, mientras que el objetivo del muestreo ascendente es recuperar información espacial. No existen limitaciones en el tamaño de la imagen. La imagen de salida tiene las mismas dimensiones que la imagen original. Se utilizan conexiones de salto para recuperar completamente la información espacial que se pierde en el muestreo descendente. Una conexión de salto es una conexión que se salta al menos una capa. En este caso se utiliza para pasar la información del paso de muestreo descendente al paso de muestreo ascendente. La fusión de características de varios niveles de resolución ayuda a combinar la información de contexto con la información espacial.

2.4.1.2. DeepLab

DeepLab (Chen *et al.*, 2018) es un modelo de código abierto de segmentación semántica de imágenes del estado del arte diseñado por Google. Este método utiliza una combinación de red neuronal convolucional profunda (DCNN) y un campo aleatorio condicional (CRF, del inglés *Connected Random Field*) utilizado habitualmente para etiquetar y segmentar secuencias de datos o para extraer información de documentos. Aborda tres grandes retos que se encuentran al aplicar una DCNN en la segmentación de imágenes: (1) Resolución reducida de las características, (2) Existencia de objetos a múltiples escalas, y (3) La reducción de la precisión de la localización resultante de la invariabilidad de la DCNN.

El modelo DeepLab aplica la convolución salteada (del inglés *Atrous convolution*) para el muestro ascendente. La combinación repetida de *max pooling* y *striding* en capas consecutivas en DCNN reduce significativamente la resolución espacial de los mapas de características resultantes. Una solución es utilizar capas de deconvolución para aumentar la muestra del mapa resultante. Sin embargo, se requiere más memoria y tiempo. La convolución salteada ofrece una alternativa sencilla y potente al uso de la deconvolución, ya que permite ampliar eficazmente el campo de visión de los filtros sin aumentar el número de parámetros ni la cantidad de cálculos.

La convolución salteada es una abreviatura de la convolución con filtros sobremuestreados. El muestreo ascendente de los filtros consiste en insertar agujeros entre las

tomas de filtro que no son cero.

En el modelo DeepLabv1 se toman las imágenes como entrada y pasa a través de capas habituales de una red neuronal convolucional profunda, seguidas de una o dos capas de convolución salteada, y obteniendo como resultado un mapa de puntuación grueso. A continuación, este mapa se muestrea al tamaño original de la imagen mediante interpolación bilineal. Por último, para mejorar el resultado de la segmentación, se aplica un CRF totalmente conectado. La Figura 12 muestra el flujo que utiliza el modelo DeepLabv1 para segmentar imágenes.

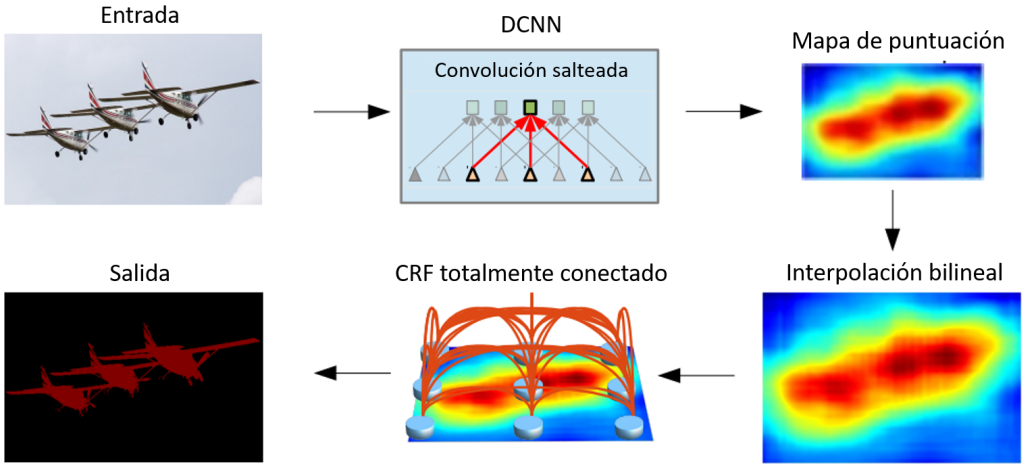


Figura 12. Flujo de la arquitectura de DeepLabv1 (Chen *et al.*, 2018).

El modelo DeepLabv2 aumenta el rendimiento de la arquitectura de DeepLabv1 abordando el reto en el que pueden existir objetos a múltiples escalas. Una forma estándar para representar un objeto en múltiples escalas es presentar a la DCNN versiones re-escaladas de una misma imagen y luego agregar los mapas de características o puntuación. Sin embargo, el modelo DeepLabv2 utiliza la técnica de Combinación salteada de Pirámides Espaciales (ASPP, del inglés *Atrous Spatial Pyramid Pooling*). La idea detrás de la combinación salteada de pirámides espaciales es aplicar al mapa de características de entrada una convolución salteada múltiple con diferentes frecuencias de muestreo y fusionarlas. Los objetos de la misma clase pueden tener diferentes escalas en la imagen, sin embargo, la técnica ASPP ayuda a tener en cuenta las diferentes escalas de los objetos, lo que ayuda a mejorar la precisión en la segmentación.

Las versiones anteriores de DeepLab son capaces de codificar información contex-

tual multiescalar averiguando los rasgos entrantes con filtros u operaciones de agrupación mediante la convolución salteada. La arquitectura de DeepLabv3 adopta un novedoso codificador-decodificador con convolución separable salteada para capturar límites de objetos más nítidos. La convolución separable salteada se aplica para aumentar la eficiencia computacional. Esto se consigue factorizando una convolución estándar en una convolución en profundidad seguida de una convolución en puntos (es decir, una convolución 1x1). En concreto, la convolución en profundidad realiza una convolución espacial independiente para cada canal de la imagen de entrada, mientras que la convolución puntual se emplea para combinar la salida de la convolución en profundidad.

2.4.1.3. Mask R-CNN

Mask R-CNN (He *et al.*, 2017) es una extensión de la arquitectura Faster R-CNN (Véase la sección 2.3.2) utilizada en detección de objetos. Faster R-CNN tiene dos salidas para cada objeto, una etiqueta de clase y un recuadro delimitador; a esto se le agrega una tercera salida que es la máscara del objeto, que es una máscara binaria que indica los píxeles en los que se encuentra el objeto en el recuadro delimitador. La salida de la máscara es distinta de las salidas de clase y recuadro delimitador, por lo que se requiere la extracción de una manera más fina del espacio que ocupa el objeto. Para ello, se utiliza la red de convolución completa (FCN).

La función de pérdida del modelo es la pérdida total al realizar la clasificación, generar el recuadro delimitador y generar la máscara del objeto.

En resumen, la arquitectura de Mask R-CNN combina las dos redes: Faster R-CNN y FCN para construir su arquitectura. En la Figura 13 puede visualizar la arquitectura de Mask R-CNN.

2.5. k-vecinos más cercanos

El algoritmo «*k* vecinos más cercanos» (kNN) es uno de los algoritmos más utilizados en aprendizaje automático, es ampliamente utilizado en tareas de clasificación

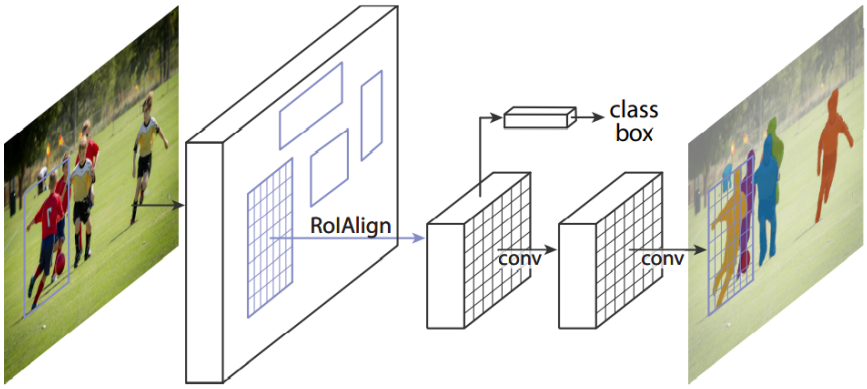


Figura 13. Arquitectura de Mask R-CNN. Esta se encuentra compuesta por Faster R-CNN (encerrada en color verde) y una red convolucional completa (encerrada en color anarajando) (He *et al.*, 2017).

y regresión. Este algoritmo corresponde al grupo de aprendizaje supervisado, es decir, los datos de entrenamiento se encuentran etiquetados con el resultado esperado, mejor conocido como «clase» o «etiqueta». La popularidad de este algoritmo se debe principalmente a su simplicidad y efectividad. A pesar de su simplicidad, kNN puede lograr resultados altamente competitivos e incluso superar a otros clasificadores más potentes. kNN puede ser utilizado para el reconocimiento de patrones, sistemas de recomendación, extracción de datos, detección de anomalías, búsqueda semántica, minería de datos, entre otros.

La idea detrás de kNN es sencilla: el algoritmo calcula la distancia entre una consulta y cada uno de los objetos en el conjunto de entrenamiento, o base de datos, para obtener la vecindad conformada por los k vecinos más cercanos. Es necesario definir previamente una función para calcular la distancia y el valor de k. El valor de k es un hiperparámetro que representa los objetos más cercanos, según la función de distancia utilizada, a la consulta. Para medir la distancia entre objetos es necesario crear o generar una representación vectorial de cada objeto. Entre las métricas más populares para «medir la cercanía» entre objetos se encuentra la distancia euclidiana y la similitud de coseno, sin embargo, existen otras métricas que pueden utilizarse como Mahalanobis. En problemas de clasificación, para determinar la etiqueta de una consulta que ha sido clasificada mediante kNN, se suele asignar la etiqueta que más se repite en los k vecinos más cercanos.

En contraste con otros algoritmos de aprendizaje supervisado, kNN no aprende una

función discriminatoria a partir de los datos del conjunto de entrenamiento, sino que el aprendizaje ocurre cada vez que se realiza una consulta. Es por ello que kNN requiere almacenar todo el conjunto de entrenamiento para clasificar una nueva consulta. A esto se le conoce como aprendizaje ocioso (del inglés *lazy learning*).

Los resultados obtenidos con kNN son muy sensibles a:

- El **hiperparámetro** k, de modo que con valores distintos de k se obtendrán resultados muy distintos. Este valor se elige después de un proceso de pruebas.
- La **función de distancia** utilizada, ya que esta influirá en las relaciones de cercanía de los puntos que se van estableciendo en el proceso de construcción del algoritmo. En la Figura 14 se muestra un ejemplo de este fenómeno. Se puede visualizar que al clasificar una consulta la etiqueta varia con respecto a la elección del hiperparámetro k.

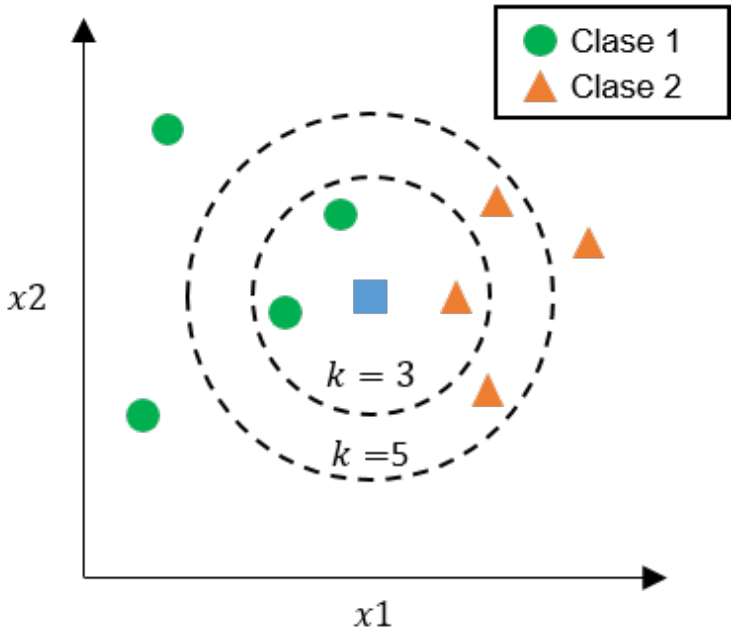


Figura 14. Clasificación de una consulta con kNN. Se muestra la clasificación de una consulta (cuadrado azul) y la etiqueta asignada con respecto al valor del hiperparámetro k. Si se selecciona una $k = 3$, la etiqueta será un círculo verde. Mientras que si el valor de $k = 5$, la etiqueta será un triángulo naranja. Lo anterior demuestra la sensibilidad que tiene la elección del hiperparámetro k.

Las fortalezas de kNN son su simplicidad, efectividad y que no requiere conocer la distribución de los datos en general. Por otro lado, sus principales desventajas son la

elección de la función de distancia, alto consumo de memoria, complejidad computacional lineal, y la estimación del hiperparámetro k.

2.5.1. Búsqueda aproximada del vecino más cercano

La implementación del algoritmo kNN es muy simple, sin embargo, la complejidad computacional de este enfoque aumenta linealmente con el número de elementos almacenados, lo que lo hace inviable para conjunto de datos a gran escala. Esto ha provocado un gran interés en el desarrollo de algoritmos kNN rápidos y escalables.

Los algoritmos de búsqueda aproximada del vecino más cercano (ANN, del inglés *Approximate nearest neighbor*) tienen como objetivo reducir la complejidad computacional y obtener exactitudes altas al momento de buscar los vecinos más cercanos de una consulta. Estos métodos por lo general, se usan como un etapa fuera de línea en el algoritmo kNN para acelerar las consultas (Andoni *et al.*, 2018).

Las soluciones más populares de ANN son los algoritmos de árbol (Muja y Lowe, 2014), el hashing sensible a la localidad (del inglés *Locality-sensitive hashing*) (Indyk y Motwani, 1998), y la cuantificación del producto (del inglés *Product quantization*) (Jégou *et al.*, 2011). Los algoritmos basados en grafos de proximidad han ganado recientemente popularidad al ofrecer un mejor rendimiento en conjuntos de datos de alta dimensión.

El grafo *Navigable Small World* (NSW) propuesto por Malkov *et al.* (2014) es uno de los grafos más eficientes para la búsqueda aproximada del vecino más cercano. Cada vez que se inserta un nuevo elemento, se encuentran los vecinos aproximados del elemento mediante una búsqueda voraz dentro del grafo parcialmente construido. El grafo *Hierarchical Navigable Small World* (HNSW) propuesto por Malkov y Yashunin (2020) es una extensión del NSW. Este grafo construye de forma incremental una estructura multicapa que consta de un conjunto jerárquico de grafos de proximidad (capas) para subconjuntos anidados de los elementos almacenados. El grafo HNSW es uno de los índices probabilísticos más eficientes para ANN hasta el momento (Li *et al.*, 2020).

2.6. Grafo HSP

Un grafo es una estructura de datos compuesta por un conjunto de objetos conocidos como nodos que se relacionan con otros nodos a través de un conjunto de conexiones conocidas como aristas. Estos permiten representar interrelaciones entre objetos que interactúan entre si. Para definir un grafo G se usa la relación $G = (V, E)$, donde cada arista $e \in E$ se asocia a un conjunto de nodos o vértices $u, v \in V$. Si $e \in E$ se encuentra asociada a u, v se dice que u y v son vecinos, o que son adyacentes.

Existen distintos tipos de grafos que se definen a partir de sus propiedades y características. Los grafos de proximidad son grafos geométricos, construidos sobre un conjunto finito S de puntos en el plano, donde la vecindad de cada vértice se define a partir de cierta noción de proximidad. Los grafos geométricos son aquellos donde los vértices representan puntos en cualquier posición en el plano y las aristas se trazan como segmentos de línea recta.

El grafo HSP diseñado por Chávez *et al.* (2005) es un grafo de proximidad local que fue propuesto para aplicaciones en redes ad hoc. Estas redes son un tipo de red inalámbrica descentralizada compuesta por transmisores, llamados *hosts*, que se establecen sin una infraestructura pre-existente. Por lo general, se representan estas redes utilizando grafos del discos unitario (UDG, del inglés *Unit Disk Graphs*). Estos grafos se forman a partir de una colección de puntos en el plano euclidiano, dos puntos son conectados si su distancia se encuentra por debajo de un umbral fijo.

El algoritmo del grafo HSP determina qué vecinos se retienen dentro del rango de cada nodo para construir un subgrafo geométrico adecuado del UDG. El grafo resultante, denominado grafo HSP, es un subgrafo disperso dirigido o no dirigido del UDG.

En la Figura 15 se muestra la selección de vecinos para un nodo arbitrario utilizando el grafo HSP. El área sombreada representa los semi-espacios que se agregan iterativamente a la región prohibida. Una arista (u, z) está prohibida por una arista (u, v) cuando la distancia euclidiana de z a v es menor que la distancia euclidiana de z a u. Es importante ver que cada nodo elige a sus vecinos sin necesidad de parámetros.

El grafo se puede construir de manera completamente distribuida y es computacionalmente simple. El algoritmo lo ejecuta cada nodo usando solo información de su

vecindario. Una característica relevante del HSP es que proporciona diversidad y similitud en la vecindad de cada nodo. La vecindad de un nodo comprende a los vecinos después de eliminar la redundancia entre ellos.

2.7. Resumen del capítulo

A lo largo de este capítulo se presentaron conceptos relacionados a la recuperación de imágenes. Las redes neuronales se han convertido en el estándar para extraer características de una imagen. Además, el algoritmo kNN debido a su simplicidad, es una manera fácil y robusta de recuperar objetos de una colección. Por último, se presentó el grafo HSP que se utilizará en esta tesis como algoritmo para recuperar imágenes.

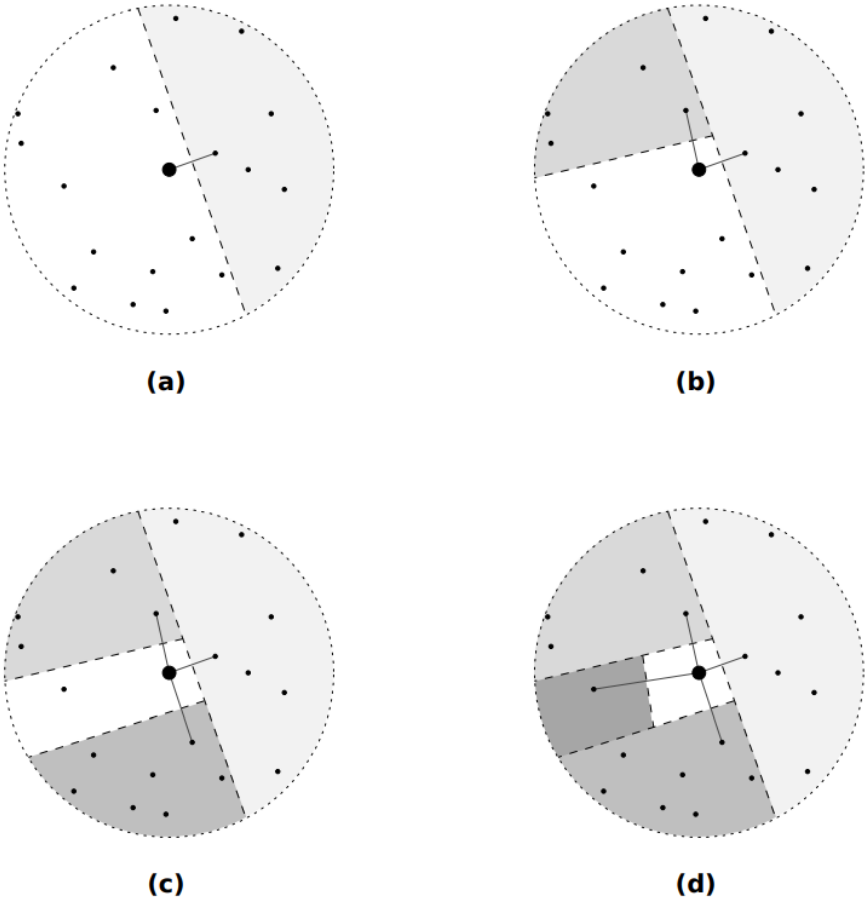


Figura 15. En (a) el nodo arbitrario u selecciona a su vecino más cercano v, es decir, el nodo con menor distancia euclidiana, trazando una arista (u, v). El área sombreada representa la región prohibida. Una arista (u, z) está prohibida por una arista (u, v) cuando la distancia euclidiana de z a v es menor que la distancia euclidiana de z a u. En (b) el nodo arbitrario u selecciona al vecino más cercano que no se encuentra en la región prohibida. En (c) el nodo arbitrario u selecciona al tercer vecino más cercano que no está en la región prohibida. Por último, en (d) el nodo arbitrario u selecciona al cuarto y último vecino más cercano que no está en la región prohibida. En total, el nodo arbitrario u tiene cuatro vecinos relevantes, los cuales se obtienen al construir el grafo HSP.

Capítulo 3. Sistemas de recuperación multimedia

Los sistemas de recuperación multimedia cada vez incluyen una mayor cantidad de módulos para recuperar un objeto específico. Esencialmente, los sistemas contienen un proceso fuera de línea, es decir, que se realiza sin la intervención de internet. Este proceso sirve para etiquetar una colección multimedia. Por otra parte, cuentan con un proceso en línea que permite interactuar con el sistema. Generalmente, se compone de un *backend* que contiene toda la lógica del sistema y un *frontend* para que el usuario interactúe con los diferentes módulos (Véase Figura 16).

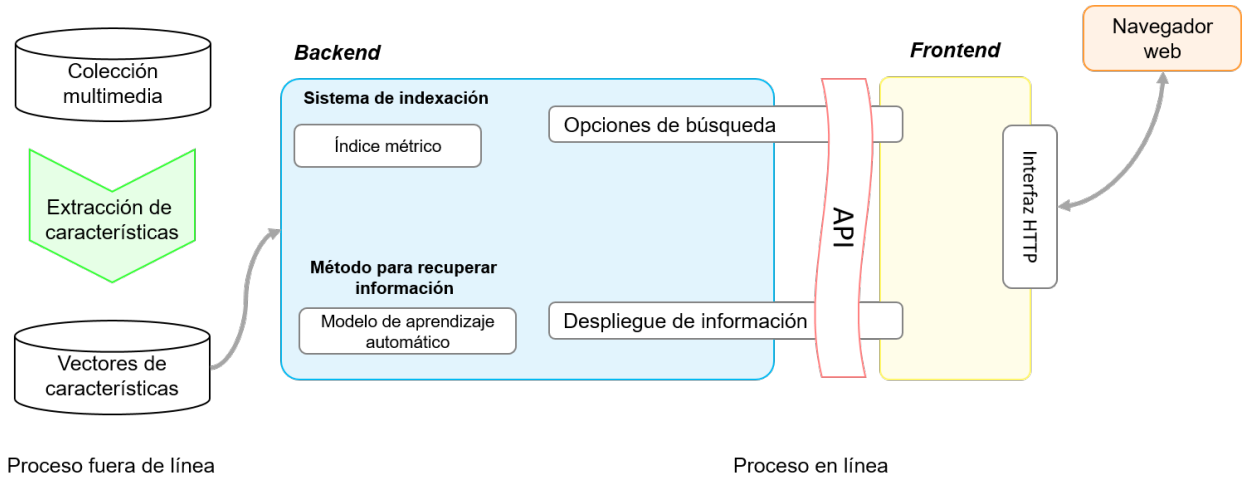


Figura 16. Arquitectura general de un sistema de navegación multimedia. Se pueden identificar tres partes: (1) Etapa fuera de línea: en donde se utilizan modelos de aprendizaje profundo para extraer características, (2) *Backend*: se conforma de un sistema de indexación, un modelo para presentar los elementos relevantes y las opciones de búsqueda, y (3) *Frontend*: la interfaz de usuario que permite interactuar con la colección multimedia.

La investigación en sistemas de recuperación multimedia ha aumentado durante la última década, gracias al crecimiento en investigación de aprendizaje profundo. Los modelos de aprendizaje profundo permiten realizar etiquetado en imágenes y video. Esto ha permitido investigar nuevos algoritmos de navegación y reducir el tiempo dedicado en el proceso de etiquetado, el cual tiempo atrás era un proceso bastante complicado y que requería mucho tiempo.

En este capítulo se presentarán algunos sistemas con características semejantes al prototipo que se pretende desarrollar. Además, se establecerán los conceptos básicos necesarios para realizar el desarrollo web de un sistema. También se introduce la utilidad de emplear la plataforma de Google Cloud y algunas herramientas que son útiles

para construir un sistema y aprovechar los beneficios del uso de la nube.

3.1. Sistemas del estado del arte

3.1.1. SOM Hunter

SOM Hunter [1] es un motor interactivo de recuperación de video. Este sistema utiliza un tipo de aprendizaje no supervisado conocido como SOM (del inglés *self-organizing maps*). SOM es un algoritmo de agrupamiento que permite producir una representación discretizada en baja dimensión de nubes de puntos de alta dimensión. La Figura 17 presenta el proceso de búsqueda de este sistema. El proceso puede iniciar opcionalmente mediante una búsqueda de texto (específicamente por palabras clave), esto tiene como objetivo acotar el conjunto inicial de datos (Kratochvíl *et al.*, 2020) y devolver un conjunto de datos ordenado por importancia o relevancia con una distribución uniforme de relevancia. Para desplegar los resultados se utiliza un SOM entrenado con características profundas. En el sistema SOM-Hunter, las pantallas organizadas por SOM son producidas de la siguiente manera: Se entrena la rejilla SOM en un espacio de características profundas, los nodos de la rejilla son utilizados como un clasificador de vecinos más cercanos y finalmente, por cada grupo de cuadros con la misma clasificación es seleccionado un cuadro como representante del grupo, el cual será desplegado en la rejilla SOM (Kratochvíl *et al.*, 2020). El despliegue mediante SOM permite utilizar aprendizaje interactivo para refinar la búsqueda del usuario.

El sistema SOM Hunter se compone de un *backend* implementado en C++ para un uso eficiente de la memoria, rapidez en los cálculos y fácil acceso a los datos indexados. Para realizar el *backend* se implementó una interfaz de programación de aplicaciones (conocida también por las siglas API, del inglés *Application Programming Interface*) para acceder a recursos mediante llamadas del protocolo de transferencia de hipertexto (del inglés *Hypertext Transfer Protocol*, abreviado HTTP). Esta API proporciona la información requerida a utilizarse en la interfaz de usuario como top-k videos, detalle de vídeos, vecinos más cercanos de un video y las pantallas SOM. La interfaz de usuario o *frontend* (Véase la Figura 18) fue implementada utilizando la tecnología Node.js por medio de un servidor. El *frontend* se compone de: (1) Una caja de texto

[1] Código fuente disponible en `https://github.com/siret/somhunter`

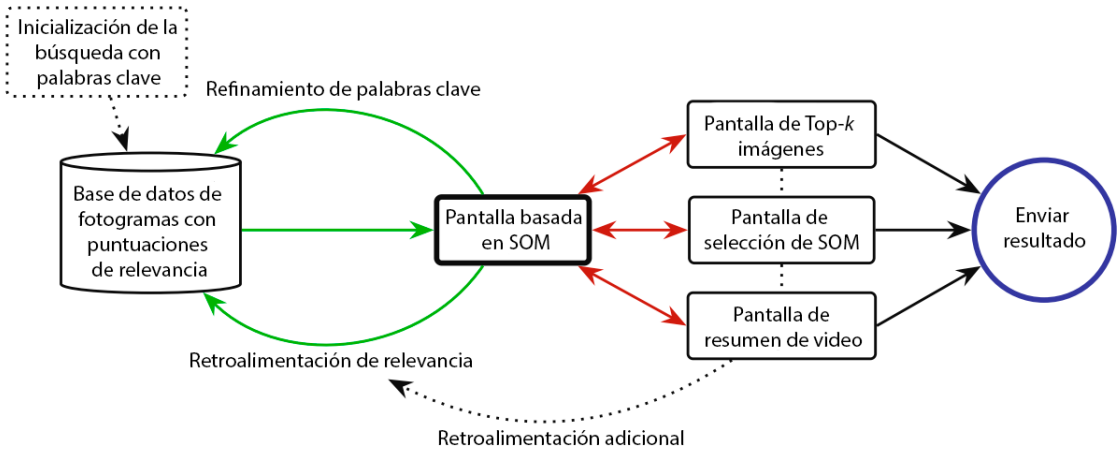


Figura 17. Descripción general del flujo de búsqueda típico en SOM-Hunter (Kratochvíl *et al.*, 2020).

para realizar consultas por palabras clave, (2) Marcas en los elementos para realizar aprendizaje interactivo, (3) Opciones para cambiar la vista de las imágenes y (4) Varios botones de diferentes acciones, tales como reiniciar la búsqueda, mostrar los vecinos más cercanos de un elemento, envío de resultados, entre otros.

3.1.2. Exquisitor

Exquisitor es un navegador multimedia enfocado en el manejo de bases de datos de gran escala (Ragnarsdóttir *et al.*, 2019). Para probar su funcionamiento se indexó la colección YFCC100M (Thomee *et al.*, 2016) que contiene 99.2 millones de imágenes y 800 mil videos. Este sistema es capaz de recuperar elementos relevantes en menos de 30 ms con recursos de cómputo limitados. Su eficiencia se debe a la representación, compresión, e indexación de los datos.

La interfaz de usuario de Exquisitor se muestra en la Figura 19. Esta interfaz permite al usuario etiquetar imágenes positivas (imágenes relevantes para el usuario) y negativas (imágenes no relevantes para el usuario). Cada vez que el usuario etiqueta imágenes estas se utilizan para mostrar nuevas sugerencias y realizar un nuevo ciclo de retroalimentación.

Para permitir que el usuario interactúe con las imágenes de la colección YFCC100M, se extraen características visuales y de texto de cada imagen. Para características visuales, fueron extraídos 1000 conceptos semánticos del dataset ImageNet utilizando

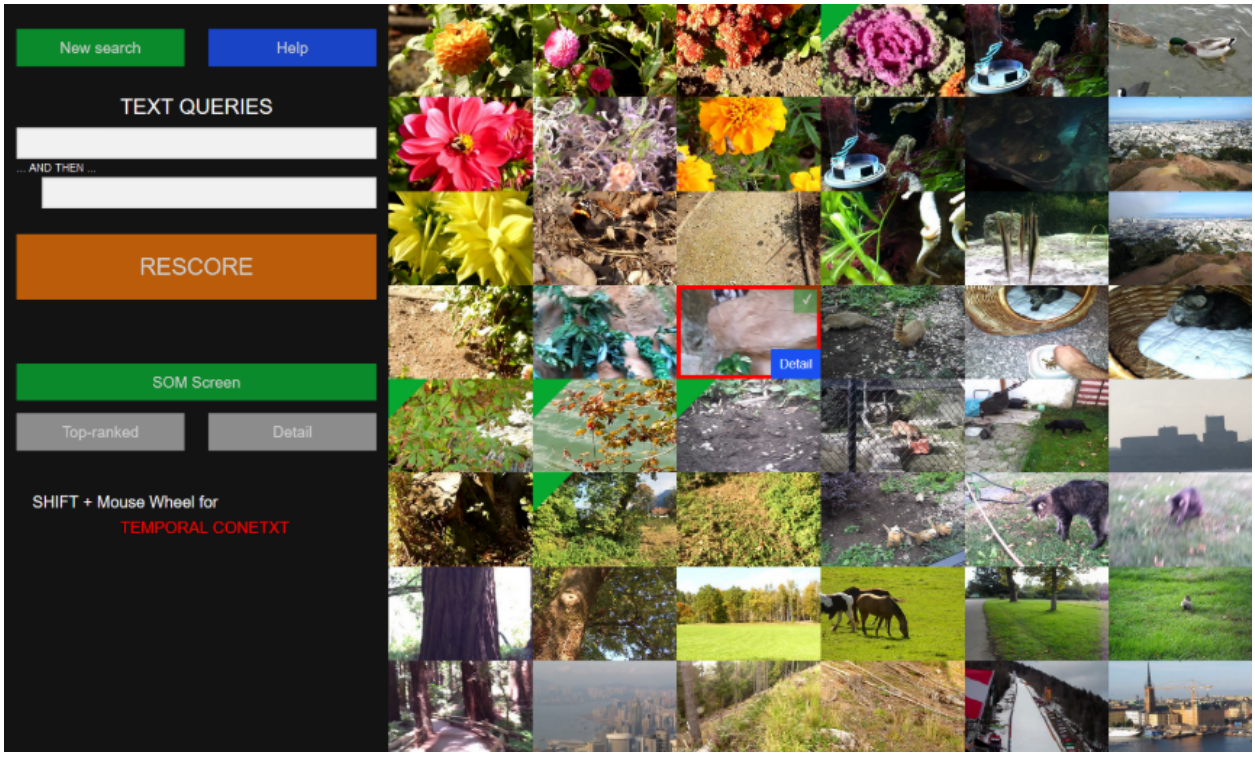


Figura 18. Interfaz de usuario de SOM-Hunter (Kratochvil *et al.*, 2020).

la arquitectura GoogLeNet. Para características de texto, se extrajeron 100 temas de Análisis Discriminante Lineal (ADL, o LDA por sus siglas en inglés, *Linear Discriminant Analysis*) del título de la imagen, las etiquetas y la descripción.

Las características visuales y de texto extraídas de las 99.2 millones de imágenes requieren casi 800 GB de memoria. Sin embargo, la representación Ratio64 (Zahálka *et al.*, 2018) permite comprimir las características a menos de 6GB de memoria con solo preservar los principales conceptos visuales y temas de texto de cada imagen. Estas características comprimidas se indexan utilizando una variante del algoritmo de indexación de alta dimensionalidad de Clúster Pruning extendido (eCP, del inglés *extended Cluster Pruning*). Además, se utiliza un modelo lineal de máquinas de soporte vectorial (SVM, del inglés *Support Vector Machines*) que facilita el funcionamiento de aprendizaje interactivo para presentar nuevos elementos relevantes después de realizar el etiquetado de imágenes (Zahálka *et al.*, 2018). La arquitectura descrita anteriormente se muestra en la Figura 20 en donde Exquisitor fue aplicado al dominio de colecciones de video.

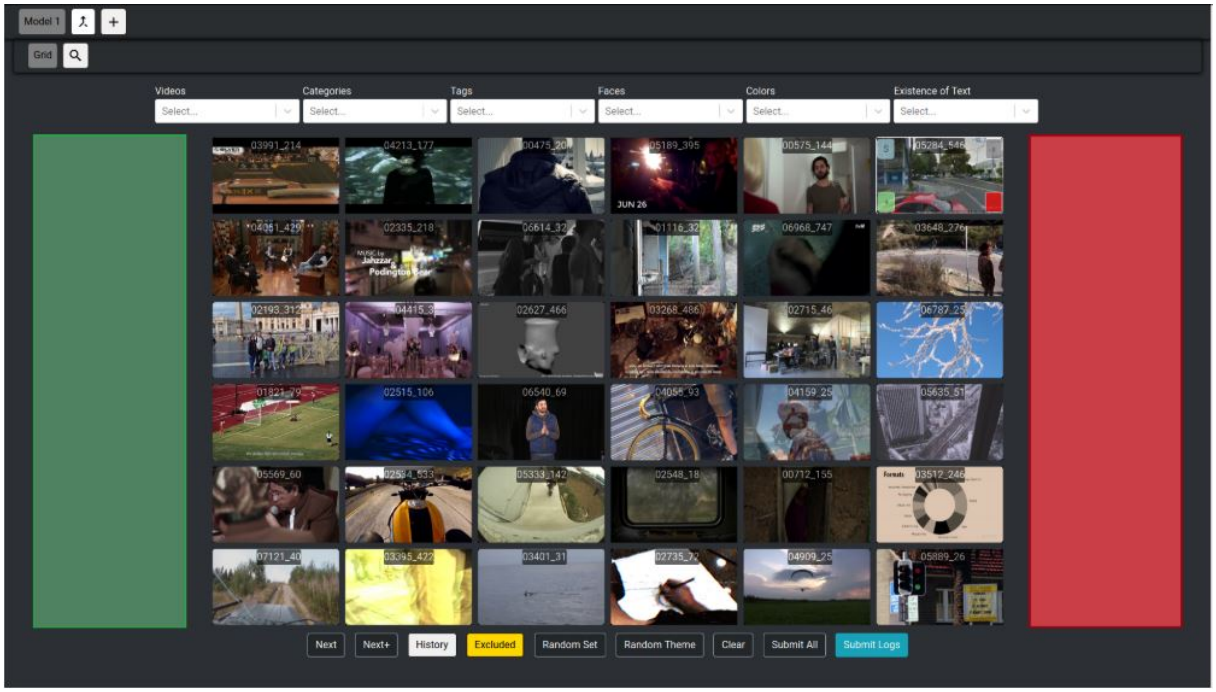


Figura 19. Interfaz de usuario de Exquisitor. La columna verde (izquierda) permite etiquetar imágenes relevantes y la columna roja (derecha) permite etiquetar imágenes como no relevantes. En el centro de la interfaz se presentan los resultados en una rejilla de 6x6 (Khan *et al.*, 2021).

3.1.3. VIRET

VIRET (Lokoč *et al.*, 2020) es un sistema desarrollado por el mismo equipo de investigación que SOM-Hunter, teniendo la similitud en que utilizan la misma representación de imágenes para la búsqueda por texto; además, permite crear consultas temporales y combinaciones de búsqueda basada en texto. En la última iteración de VBS, implementaron el modelo BERT (Devlin *et al.*, 2019) para enriquecer las búsquedas por texto. BERT es un modelo complejo y de buen rendimiento que permite a los usuarios realizar una descripción de texto con mucho detalle para realizar búsquedas.

3.1.4. VERGE

VERGE es un motor de búsqueda interactivo que permite navegar en colecciones multimedia de imágenes y videos. Su interfaz integra una amplia gama de módulos para recuperar objetos. Este sistema permite al usuario realizar consultas mediante recuperación basada en conceptos, búsqueda de similitud, coincidencia de texto con video, detección de rostros, búsqueda basada en los subtítulos y reconocimiento de

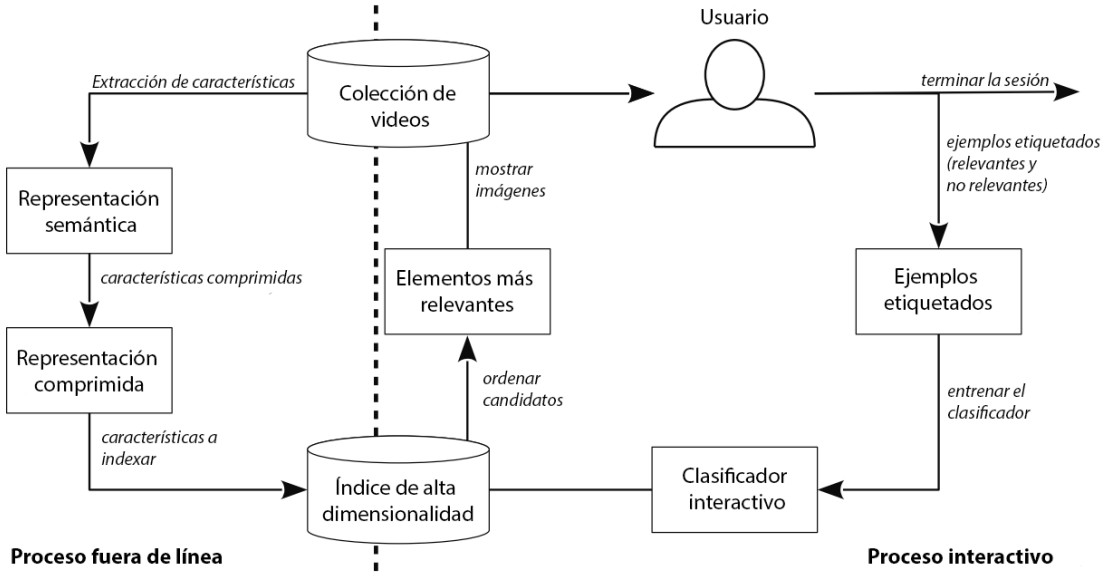


Figura 20. Arquitectura de Exquisitor para aprendizaje interactivo. Inicialmente, se extraen características y se realiza una representación comprimida para indexarlas en un índice de alta dimensionalidad; este proceso se realiza offline. En proceso interactivo, se presentan posibles videos relevantes al usuario. El usuario etiqueta videos como relevantes y no relevantes y esta retroalimentación es utilizada para entrenar un clasificador basado en máquinas de soporte vectorial, que permite recuperar un nuevo conjunto de videos a presentar al usuario (Jónsson *et al.*, 2020).

actividad (Peška *et al.*, 2021).

Dentro del estado del arte de sistemas de navegación multimedia, existe una gran variedad en el diseño de su arquitectura, herramientas de consulta, y modelos de aprendizaje profundo para el procesamiento de los datos. Cada sistema cuenta con su particularidad, lo que permite la exploración de diferentes enfoques. La variedad que existe es tanta, que inclusive en el diseño de interfaz de usuario se ha llegado a explorar el uso de realidad aumentada (Duane *et al.*, 2020).

3.1.5. Sistemas de recuperación multimedia comerciales

En el ámbito comercial también existen sistemas que son relevantes mencionar. Pinterest[2] y Google Images[3] son dos ejemplos de sistemas de navegación multimedia que están diseñados para recuperar imágenes.

Pinterest (Véase Figura 21) es una red social, que esencialmente funciona como un

[2] https://www.pinterest.com.mx/
[3] https://www.google.com/imghp

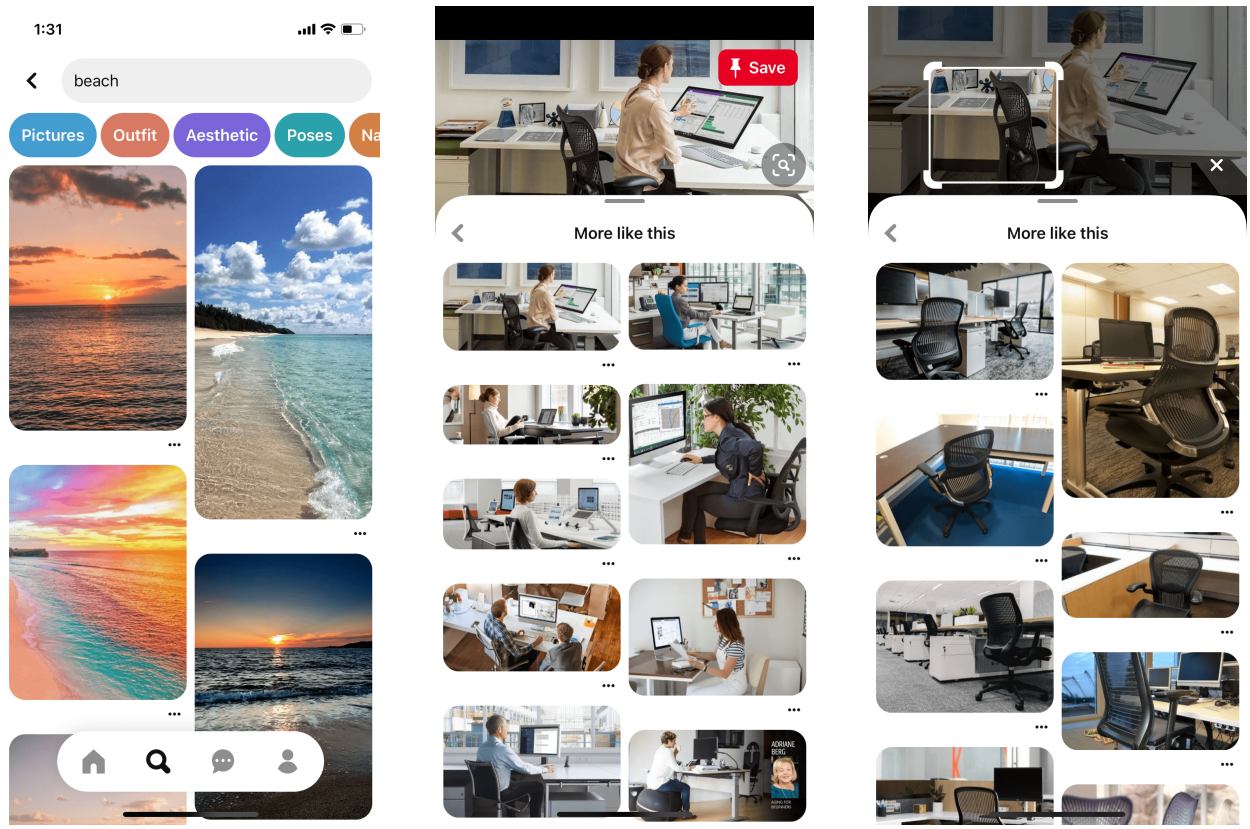
navegador multimedia de imágenes. Esta herramienta permite a los usuarios buscar imágenes (conocidos como pines, del inglés *pins*) y agregarlos a tableros (del inglés *boards*) que por lo general, los tableros se encuentran organizados por un mismo contenido temático. Pinterest permite realizar búsquedas basadas en texto, subiendo una imagen o seleccionando un objeto de una imagen. También es posible realizar búsquedas de semejanza, también conocido como sistema de recomendación, cuando es seleccionado un pin. Pinterest utiliza redes neuronales convolucionales para extraer características profundas y detección de objetos en imágenes (Zhai *et al.* (2017); Jing *et al.* (2015).

Google Images es una especialización del buscador de Google para búsqueda de imágenes. Las búsquedas pueden ser iniciadas mediante una consulta de texto, un enlace de una imagen o un archivo de una imagen. Después de realizar una búsqueda, se muestran los resultados ordenados por relevancia en la interfaz de usuario. El algoritmo permite asociar imágenes entre sí para crear grupos (o del inglés *clusters*) al momento de seleccionar una imagen y proporcionar una función de búsqueda de imágenes inversa. Además, permite aplicar filtros de tamaño, color, tiempo, formato de imagen (artístico, dibujo a mano o formato de intercambio de gráficos (más conocido por las siglas GIF, del inglés *Graphics Interchange Format*)) y etiquetas similares a la consulta.

TinEye [4] es una herramienta comercial que contrasta con los sistemas mencionados anteriormente. TinEye es un motor de búsqueda inversa de imágenes. No es un navegador multimedia como tal, ya que no permite realizar exploración y es utilizado para conocer de dónde proviene cierta imagen, proporcionando un localizador de recursos uniforme (más conocido por las siglas URL, del inglés *Uniform Resource Locator*) de la imagen o subiendo el archivo de la imagen. Los resultados se presentan mediante un listado con páginas web donde se encuentra la imagen.

TinEye, Google Images y Pinterest son sistemas muy utilizados en la web. Sin embargo, sus representaciones internas no son públicas. No obstante, esto es diferente con los sistemas del estado del arte como SOMHunter, Exquisitor, VIRET o VIREO, entre otros; en estos últimos sí conocemos con mucho detalle sus características internas. Una característica en común es el uso de redes neuronales para extraer características

[4] `https://tineye.com/`



(a) (b) (c)

Figura 21. Interfaz de usuario de Pinterest. (a) Muestra el resultado de la búsqueda de texto de "beach"(playa). (b) Presenta los resultados de una búsqueda dado un ejemplo, y en (c) Se selecciona una parte de la imagen (silla) utilizada en (b) para realizar una búsqueda dado un ejemplo.

de objetos multimedia (por ejemplo, imágenes o videos).

3.2. Desarrollo Web

El desarrollo es la creación y mantenimiento de sitios web utilizando varios lenguajes de programación. Su principal tarea es la creación de una interfaz gráfica para que el usuario pueda interactuar con el sistema o sitio web. Esta interfaz gráfica debe cumplir con requerimientos, como buen aspecto, funcionamiento, rendimiento, y experiencia de usuario.

El campo de desarrollo web suele dividirse en *frontend* (dedicado a la interfaz gráfica, parte con la que el usuario interactúa) y *backend* (dedicado a la lógica, usualmente conocida como el servidor).

3.2.1. Desarrollo web frontend

El *frontend* (Godbolt, 2016) es la parte del desarrollo web encargada de convertir datos en una interfaz gráfica para que el usuario interactúe con información. Generalmente el código de la aplicación se ejecuta en el navegador del usuario, al que se le denomina una aplicación cliente. Dentro del área se trabaja con lenguajes mayormente del lado del cliente, como:

- Lenguaje de marcado de hipertexto, HTML (por su nombre en inglés, *Hyper Text Markup Language*): es un lenguaje de marcado y etiquetado, sirve para realizar la estructura básica de un sitio web y el contenido a nivel de: textos, imágenes, etc. Es el estándar principal aceptado por la *World Wide Web* (WWW).
- Hojas de estilo en cascada, CSS (por su nombre en inglés, *Cascading Style Sheets*): controla el formato y diseño visual de los sitios web.
- JavaScript: Es un lenguaje de programación imperativo basado en eventos que permite darle dinamismo a una página HTML estática.

Cabe mencionar que HTML y CSS son lenguajes de marco y estilo. Mientras que JavaScript es un lenguaje de programación. Además, las tres tecnologías mencionadas anteriormente, son el estándar actual en la creación de sitios web.

3.2.1.1. Frameworks y bibliotecas

De HTML, CSS y JavaScript se genera una gran cantidad de frameworks y bibliotecas que aumentan las capacidades para generar cualquier tipo de interfaz de usuario. Actualmente existe una amplia variedad de tecnologías que se utilizan para la construcción de sistemas o sitios web. Según datos de Stack Overflow[5], los frameworks y bibliotecas más utilizados para desarrollo web son: React.js, Angular y Vue.js. React.js es una biblioteca de JavaScript desarrollada por Facebook y lanzada en el año 2013. Esta biblioteca permite crear interfaces de usuario. Por otro lado, Angular es un framework de JavaScript de código abierto desarrollado por Google. Este framework permite

[5]Encuesta de desarrolladores de Stack Overflow (2021). Recuperado el 2 de septiembre de 2021 de: https://insights.stackoverflow.com/survey/2021

desarrollar aplicaciones de página única (SPA, del inglés *Single Page Application*). La primera versión de Angular se lanzó en el 2010 con el nombre AngularJS, pero en 2016 Google decidió darle un nuevo giro a su framework y lanzar la primera versión de Angular. Vue.js es un proyecto muy pequeño a comparación de los dos anteriores, creado por Evan You con el apoyo de miembros y desarrolladores de la comunidad. La visión de Evan You, que anteriormente trabajaba para Google usando AngularJS, era extraer la parte que más le gustaba de Angular y construir algo realmente liviano. La primera versión estable fue lanzada en el año 2016.

Vue y React son los frameworks más adecuados cuando se requiere realizar aplicaciones ligeras, mientras que Angular funciona bien para aplicaciones más robustas. La curva de aprendizaje es un aspecto a considerar si no se tiene conocimiento previo de los frameworks. Hablando en términos de flexibilidad, React gana en este apartado con amplia diferencia, ya que es una biblioteca, y el usuario puede instalar los componentes que se ajusten a sus necesidades. Sin embargo, tanto Angular y Vue ofrecen todo lo que se necesita para el desarrollo web, pero no son muy flexibles.

En resumen, Vue y React ofrecen buen rendimiento, flexibilidad y son más adecuadas para aplicaciones livianas. Mientras que Angular es menos flexible y rinde mejor en aplicaciones grandes. React.js es el framework más popular en la actualidad, y que ofrece mejores beneficios. Es por ello que React.js fue seleccionado como el framework para realizar el desarrollo de la interfaz gráfica del sistema de recuperación de imágenes.

3.2.2. Desarrollo web backend

El *backend* (Dauzon *et al.*, 2017) es la parte del desarrollo web que se refiere al lado del servidor. Mientras que el *frontend* se ejecuta en el navegador del usuario, el *backend* procesa la información que alimenta el *frontend* con datos. Los procesos o funciones que realiza el *backend* no son visibles, pero tienen mucha importancia en el buen funcionamiento y rendimiento de una aplicación web. El *backend* se enfoca principalmente en gestionar el desarrollo de funciones que simplifiquen el proceso de desarrollo, acciones de lógica, uso de librerías del servicio, conexión con bases de datos, seguridad, escalabilidad, entre otros aspectos.

El desarrollo de *backend* se encuentra basado en API REST. La interfaz de programación de aplicaciones conocida por las siglas API, en inglés *Application Programming Interface*, es un conjunto de definiciones y protocolos que ofrece una capa de abstracción que asegura que pueda ser consumida desde cualquier sistema o cliente, sin la necesidad de saber cómo está implementada. La transferencia de estado representacional (en inglés *Representational State Transfer*) o REST es una arquitectura de software que permite usar HTTP para obtener datos o generar operaciones sin estado sobre esos datos en formatos como XML (acrónimo de Lenguaje de Marcado Extensible, del inglés *Extensible Markup Language*) y JSON (acrónimo de Notación de Objeto de JavaScript, del inglés *JavaScript Object Notation*).

Una API REST utiliza la arquitectura cliente-servidor con la gestión de solicitudes a través de HTTP. Además, la comunicación entre cliente y servidor es sin estado, lo cual implica que no se almacena información del cliente entre las solicitudes y cada una de ellas es independiente. Por esta razón, las API REST son rápidas y ligeras, y cuentan con mayor capacidad de ajuste, dando buenos resultados en el área de desarrollo de aplicaciones web y móviles.

Dentro del ámbito del *backend* existe una gran variedad de lenguajes utilizados para programar una API REST. A diferencia del *frontend*, en donde el estándar principal es el uso de HTML, CSS y JavaScript, en el desarrollo de *backend* no existe un lenguaje principal. Entre los lenguajes de programación con los que es posible programar se encuentran: Java, Python, JavaScript, C++, C#, PHP Ruby, Kotlin, Scala, entre otros. Cada uno de estos lenguajes tiene sus ventajas y desventajas que son necesarias tomar en cuenta dependiendo la aplicación que se pretende desarrollar. Además, cada uno de estos lenguajes de programación mencionados tiene diferentes frameworks para el desarrollo de *backend*; lo que provoca que la lista sea aún más extensa.

3.2.2.1. Frameworks

Nos enfocaremos únicamente en hablar de frameworks en los que se pueda programar con Python. Esto debido a que la mayoría de las implementaciones de redes neuronales se encuentran programadas en Python. De manera que sea posible integrar los modelos con la lógica del sistema.

Django es un framework *open source* de Python que permite desarrollar aplicaciones complejas de manera rápida y sencilla. Este framework es adecuado para sitios web basados en bases de datos sofisticadas y con una amplia gama de funcionalidades. Las ventajas que Django proporciona a los desarrolladores es su facilidad y una curva de aprendizaje moderada. Este framework fue diseñado para acelerar el proceso de desarrollo de APIs. También proporciona una amplia variedad de funcionalidades, como autenticación de usuarios, mapas de administración de contenido y muchas más. Django ofrece seguridad óptima, alta escalabilidad, y versatilidad.

Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código. Se clasifica como «minimalista» porque no requiere herramientas o bibliotecas particulares, es decir, no cuenta con capa de abstracción de base de datos, formatos de validación, o cualquier otro componente que provea librerías de terceros. Sin embargo, Flask proporciona flexibilidad de tal manera que provee soporte para instalar extensiones de terceros que agrega una funcionalidad requerida y además, se ejecuta como si estuviera implementada dentro de Flask. En comparación con Django, Flask es más adecuado para proyectos pequeños y sencillos.

Flask y Django son dos excelentes opciones para desarrollar *backend*. Sin embargo, Flask tiene una menor curva de aprendizaje y un mayor beneficio a corto plazo, dada su flexibilidad, robustez, y alto rendimiento con pocas líneas de código. Además, se encuentra enfocado al desarrollo de aplicaciones pequeñas.

3.2.2.2. Arquitecturas

Generalmente, el *backend* es construido utilizando una arquitectura monolítica. En esta arquitectura todos los procesos están asociados y se ejecutan como un solo servicio. Esto provoca que, si existe una falla dentro del servicio, toda la aplicación falle. Para resolver está problemática, existe la arquitectura de microservicios (Nadareishvili *et al.*, 2016). Los microservicios dividen el *backend* en pequeños componentes que se comunican entre sí por medio de llamadas HTTP. En caso que un microservicio falle, no ocurre una falla total y puede seguir funcionando el *backend*. La Figura 22 muestra como sería dividir un servicio que utiliza una arquitectura monolítica en microservicios.

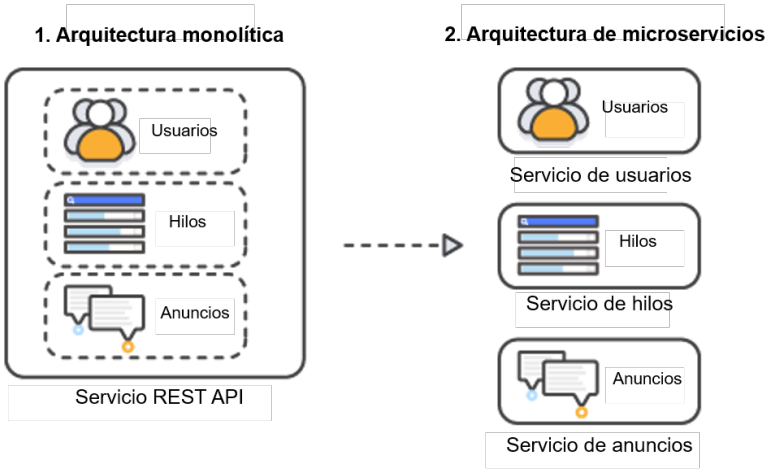


Figura 22. Dividir una aplicación monolítica en microservicios.

La ventaja de implementar una arquitectura de microservicios es que pueden ejecutarse de forma independiente, cada servicio se puede actualizar, implementar, y escalar para satisfacer la demanda específica de la aplicación.

3.3. Cómputo en la nube

La computación en la nube, más conocida como «la nube», es una tecnología que permite adquirir recursos de un sistema informático bajo demanda por medio de Internet, siendo así, una alternativa a la ejecución en una computadora o servidor local. Entre los servicios que ofrece están el almacenamiento de archivos, uso de aplicaciones o la conexión de dispositivos (Srivastava y Khan, 2018).

La nube democratiza el acceso a recursos de software de nivel internacional, ofreciendo tanto a individuos como empresas la capacidad de adquirir servicios de computación con buen mantenimiento, seguro, de fácil acceso, y bajo demanda. En el modelo de nube, no hay necesidad de instalar aplicaciones localmente en computadoras, cambiando el paradigma en que el hardware y software es diseñado y adquirido.

IaaS (Infraestructura como servicio), PaaS (Plataforma como servicio) y SaaS (Software como servicio) son los tres modelos más comunes de servicios en la nube. SaaS se centra en facilitar el acceso a una aplicación que se aloja en la nube y se accede a través de un navegador web, un cliente, o una API. Es el modelo de entrega principal

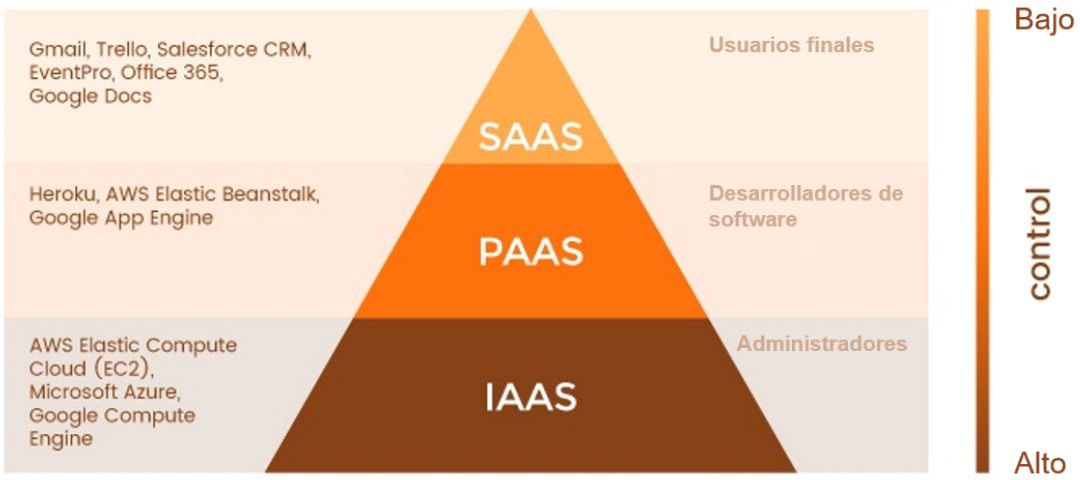


Figura 23. Los tres modelos de servicios de nube. SaaS está dirigido a usuarios finales y a su vez, tienen poco control de las aplicaciones. PaaS se encuentra dirigido a desarrolladores de software y, por último, IaaS está dirigido a administradores de tecnología, en donde se les otorga un control total sobre las herramientas.

para la mayoría de los softwares comerciales actuales. PaaS proporciona a los desarrolladores de software una plataforma bajo demanda, que incluye hardware, colección de software completa, infraestructura e incluso herramientas de desarrollo, para desarrollar aplicaciones de software personalizadas. Con PaaS, el proveedor de nube aloja servidores, redes, almacenamiento, software de sistema operativo y bases de datos en su centro de datos. Mientras que los usuarios simplemente inician los entornos que necesitan para ejecutar, desarrollar, probar, implementar, mantener, actualizar y escalar aplicaciones. Por último, IaaS proporciona a las organizaciones la capacidad de aprovechar recursos brutos del servidor mientras que el resto de la administración de la plataforma y del software es responsabilidad de la empresa. Este modelo proporciona a los usuarios el nivel más bajo de control de recursos informáticos en la nube. La Figura 23 presenta a detalle las características de los tres modelos más populares de servicios de nube.

3.3.1. Google Cloud Platform

La plataforma de Google Cloud[6] es un conjunto de servicios públicos de computación en la nube ofrecidos por Google. Estos servicios se ejecutan en la misma infraes-

[6]Google Cloud Platform (2021). Recuperado el 23 de junio de 2021 de: `https://cloud.google.com/`

tructura que Google utiliza internamente para sus productos comerciales, por ejemplo, YouTube, Google Drive, Gmail, entre otros. Sus principales competidores son Amazon Web Services (AWS) y Microsoft Azure. Cabe mencionar que en la computación en la nube, se le conoce como servicios a los productos de software y hardware.

La nube de Google se compone de recursos físicos, como computadoras y unidades de almacenamiento, y recursos virtuales, como máquinas virtuales, las cuales se encuentran en centros de datos de Google alrededor del mundo. La plataforma de Google Cloud ofrece infraestructura como servicio (IaaS), plataforma como servicio (PaaS), y entornos de computación sin servidor.

Los centros de datos de Google se encuentran ubicados en diferentes regiones, las cuales son: Asia, Australia, Europa, América del Norte, y América del Sur. Los precios entre regiones pueden variar. Además, la disponibilidad de servicios puede estar limitada a ciertas regiones. La distribución de los recursos en regiones brinda varios beneficios, principalmente la redundancia en casos de fallas y una menor latencia.

La nube de Google es capaz de distinguirse de otros proveedores de muchas formas. Como primer punto tenemos la seguridad que provee la nube de Google. El modelo de seguridad de Google es un proceso de extremo a extremo con más de 15 años de investigación y desarrollado, que además, se utiliza en sus productos comerciales. Este modelo de seguridad es el mismo que aprovecha las aplicaciones y datos en Google Cloud. El segundo punto es la facturación por segundo en donde se cobra el procesamiento por cada segundo utilizado. El tercer punto son las innovaciones tecnológicas en servicios de última generación para almacenamiento de datos en la nube (por ejemplo, BigQuery), aprendizaje automático avanzado (AI Platform), entre otros servicios. Por último, tenemos la parte ecológica. Los centros de datos de Google Cloud usan la mitad de energía de la que ocuparía un centro de datos convencional y priorizan el uso de energía renovable.

Google Cloud ofrece servicios de cómputo, almacenamiento, redes, aprendizaje automático, internet de las cosas (IoT), y *big data*, así como herramientas de gestión de la nube, seguridad y herramientas de desarrollo. La Figura 24 muestra los diferentes servicios con los que cuenta Google Cloud.

Uno de los grandes beneficios que ofrece Google Cloud es que cuenta con un pro-

Figura 24. Servicios que ofrece Google Cloud.

grama gratuito en el que otorga a los nuevos usuarios la posibilidad de probar sus productos. Para ello se les otorgan créditos gratuitos equivalentes a la cantidad de $300 dólares estadounidenses. Estos créditos pueden ser utilizados para ejecutar, probar y, además, implementar cargas de trabajo. El acceso a los productos es limitado, sin embargo, es posible acceder a más de 20 productos diferentes. Un punto importante a destacar es que el uso de GPUs no se encuentra disponible para utilizar con créditos gratuitos. Por lo tanto, los servicios que requieran una GPU no podrán ser utilizados de manera gratuita, es necesario ingresar dinero en electrónico.

En resumen, Google Cloud resulta ser una gran alternativa para entrar al mundo de la computación en la nube. La facilidad con la que se puede acceder a sus servicios es su principal ventaja para aquellos que no conocen sobre computación en la nube o simplemente, quieren realizar pruebas de distintos servicios. El beneficio de utilizar la propia infraestructura de Google dentro de nuestras aplicaciones resulta ser una gran ventaja a considerar al elegir Google Cloud por encima de otros proveedores. Además, Google Cloud es el proveedor más económico y con un catálogo de servicios en la nube muy amplio.

3.3.1.1. Compute Engine

Google Compute Engine[7] es una infraestructura como servicio que ofrece Google Cloud Platform. Este servicio de computación es seguro y personalizable con el que

[7] Google Compute Engine (2021). Recuperado el 28 de junio de 2021 de: `https://cloud.google.com/compute`

se pueden crear y ejecutar máquinas virtuales (VMs) en la infraestructura de Google. Los usuarios son capaces de lanzar máquinas virtuales según sus necesidades, seleccionando de un catálogo de máquinas virtuales predefinidas que recomienda Google o personalizar con los recursos que el usuario requiera. Entre las principales ventajas que ofrece este servicio se encuentra el escalamiento, el rendimiento, y la facilidad de crear grandes clústeres de procesamiento, todo esto acompañado del uso de la infraestructura de Google.

Las máquinas virtuales se clasifican por familia de máquinas. E2, N2, N2D, y N1 son máquinas de uso general que ofrecen un buen equilibrio entre precio y rendimiento y son adecuadas para una amplia variedad de aplicaciones, como por ejemplo, bases de datos, desarrollo web, aplicaciones móviles y web, juegos móviles y entornos de desarrollo. Es posible seleccionar hasta 224 vCPUs y 900 Gigabytes de memoria. Las máquinas virtuales M2 y M1 se caracterizan por ofrecer gran capacidad de memoria. Se pueden seleccionar hasta 12 Terabytes por instancia. Están diseñadas para cargas de trabajo que requieran gran uso de memoria, como grandes bases de datos y análisis de datos. Las máquinas optimizadas para la computación (C2) ofrecen el mayor rendimiento en cargas de trabajo que requieren grandes cantidades de recursos computacionales. Por último, las máquinas optimizadas para aceleradores (A2) integran una GPU Nvidia A100 con la arquitectura Ampere. Estas máquinas virtuales están diseñadas para trabajar con cargas de aprendizaje automático.

Los precios de Compute Engine se basan en el uso por segundo considerando el tipo de máquina, uso de disco y demás recursos con los que cuente la máquina virtual. Además, existen descuentos por uso sostenido. Estos descuentos se aplican si la máquina virtual se usa cierto porcentaje del mes.

3.3.1.2. Cloud Storage

Google Cloud Storage[8] es un servicio de almacenamiento de objetos seguro y fiable en la nube que permite almacenar cualquier tipo de archivo y acceder a ellos mediante el uso de su REST API. Este servicio aprovecha la infraestructura de Google para

[8]Google Cloud Storage (2021). Recuperado el 2 de julio de 2021 de: `https://cloud.google.com/storage`

ofrecer rendimiento, escalabilidad, encriptación de datos, capacidades avanzadas de seguridad y compartición.

Para almacenar objetos es necesario crear un bucket. Los buckets son contenedores lógicos para guardar objetos, como una carpeta raíz en un sistema operativo. El nombre del bucket funciona como una clave exclusiva que es asignada por el usuario y además, es única dentro de la infraestructura de Google. Cada bucket puede almacenar hasta 5 Terabytes. Para acceder a los objetos dentro de un bucket es necesario autorizar todas las solicitudes mediante una lista de control de acceso que está asociada a cada bucket y objeto; también es posible convertir los archivos públicos para que cualquier persona tenga acceso a ellos. El nombre de los buckets y objetos se debe elegir de modo que sean direccionables mediante las URLs de HTTP, por ejemplo «*https://storage.cloud.google.com/bucket/object*».

Es posible seleccionar el tipo de almacenamiento entre cuatro clases diferentes que son idénticas en rendimiento, latencia y durabilidad. Una clase de almacenamiento es una parte de los metadatos que usa cada objeto. La clase seleccionada afecta la disponibilidad y el precio del almacenamiento.

- **Standard Storage**: Es una clase ideal para almacenar datos que se acceden con frecuencia, como los sitios web, aplicaciones móviles y vídeos en streaming. Su costo es de 0.02 USD por GB al mes.
- **Nearline Storage**: Es una clase para almacenar datos a los que se accede con poca frecuencia. Es ideal para los datos que en promedio se leen o modifican una vez al mes o menos. Se utiliza principalmente para almacenar datos durante al menos 30 días. Su costo es de 0.01 USD por GB al mes.
- **Coldline Storage**: Se trata de una clase de coste para almacenar datos durante al menos 90 días. Es ideal para datos que son leídos o modificados una vez por trimestre. Se utiliza principalmente para almacenar copias de seguridad. Su costo es de 0.004 USD por GB al mes.
- **Archive Storage**: Es una clase ideal para almacenar datos en un periodo de tiempo mínimo de 365 días. Es la clase más barata y pensada para almacenar

copias de seguridad y recuperación de desastres. Tiene un alto costo para operaciones de acceso y lectura. Su costo es de 0.0012 USD por GB al mes.

Cloud Storage resulta ser una muy buena opción para almacenar objetos, específicamente imágenes, dado que cuenta con muchas ventajas que pueden ser bien aprovechadas en el desarrollo e implementación del sistema de navegación multimedia. La baja latencia de los objetos permite acelerar el despliegue de imágenes al realizar una búsqueda, lo cual es una gran ventaja al momento de interactuar con el sistema para recuperar una imagen.

3.4. Resumen del capítulo

La construcción de un sistema de navegación multimedia requiere conocimientos en múltiples áreas, como recuperación de información, visión por computadora, cómputo en la nube, y desarrollo web. En este capítulo se presentaron los sistemas con características semejantes al propuesto en esta tesis. Además, se introdujeron conceptos necesarios para el desarrollo web del sistema.

Capítulo 4. Metodología

Para desarrollar un sistema de recuperación de imágenes, se propone una metodología que parte del etiquetado de las imágenes. Esta metodología consiste en mejorar la representación de imágenes con características profundas utilizando modelos de aprendizaje profundo preentrenados y construir un encaje binario de imágenes, etiquetar una colección de imágenes, y utilizar la información obtenida para construir un *backend* y un *frontend*.

La mayoría de los sistemas de recuperación de información pertenecientes al estado del arte, utilizan características profundas para representar objetos multimedia (sean imágenes o videos). Algunos ejemplos de estos sistemas son: Exquisitor (Ragnarsdóttir *et al.*, 2019), SOMHunter (Kratochvíl *et al.*, 2020), VERGE (Peška *et al.*, 2021), VIRET (Lokoč *et al.*, 2020), y VIREO (Nguyen *et al.*, 2020). Sin embargo, la alta dimensionalidad de las características profundas representa todo un reto al trabajar con colecciones de gran escala, es decir, con gran número de datos. La alta dimensionalidad representa más espacio en disco duro, mayor consumo de memoria RAM, y más ciclos de cómputo al realizar operaciones.

El sistema SOMHunter (Kratochvíl *et al.*, 2020) resuelve el problema de la alta dimensión de las características profundas con un algoritmo de aprendizaje no supervisado conocido como SOM (del inglés *self-organizing maps*). Este algoritmo produce una representación discretizada en baja dimensión de las características profundas. Por otro lado, el sistema Exquisitor (Ragnarsdóttir *et al.*, 2019) utiliza la representación Ratio64 (Zahálka *et al.*, 2018) para comprimir las características profundas y preservar las principales características visuales.

Se pretende lograr una representación binaria de las imágenes de colecciones multimedia que permita construir un sistema de recuperación de imágenes. Por lo tanto, para evaluar esta metodología, se realizarán experimentos con diferentes conjuntos de datos de imágenes en tareas de navegación.

4.1. Etiquetado de imágenes

Se propone etiquetar colecciones de imágenes mediante el uso de modelos de aprendizaje profundo preentrenados. Para extraer las características profundas es necesario extraer la salida de la penúltima capa de un modelo preentrenado, por ejemplo, VGG16, ResNet50, GoogLeNet, entre otros. La representación vectorial obtenida permite usar algoritmos de aprendizaje automático (por ejemplo, kNN) para clasificarlos.

La extracción de características profundas de modelos como VGG16 o ResNet50 proporciona un vector de alta dimensión. Por ejemplo, si utilizamos la red VGG16, se obtiene un vector de 4,096 dimensiones. Mientras que al utilizar ResNet50, se obtiene un vector de 2,048 dimensiones. Considerando que los valores del vector son flotantes y se tiene una colección de millones de imágenes, se consumirá mucha memoria principal (RAM). Por ejemplo, si cada vector tiene una dimensión de 4,096 y se almacenan en formato de punto flotante de simple precisión, es decir, 32 bits o 4 bytes por cada valor, tendremos que cada vector necesitará 131,072 bits o 16,384 bytes. Supongamos que se tiene que etiquetar una colección de un millón de imágenes y se utiliza un modelo VGG16 para extraer las características profundas de cada imagen. Entonces eso quiere decir que serán requeridos 16.38 GB de memoria RAM. Esto provocará el aumento en el consumo de memoria RAM al etiquetar colecciones de imágenes de mayor tamaño, lo cual se vuelve ineficiente. Por lo anterior, se propone el uso de códigos de Hadamard para codificar la salida de las redes neuronales y obtener un vector binario de cada imagen de la colección. Este vector se utilizará en el sistema de recuperación de imágenes para realizar búsqueda dado un ejemplo y retroalimentación positiva.

Además, se utilizaron modelos de segmentación panóptica que permiten identificar de una imagen cosas y objetos. Esto permite realizar búsqueda por texto en un sistema de recuperación de imágenes. Cabe mencionar que este apartado no será evaluado por cuestión de tiempo, solamente se etiquetará la colección para integrar el módulo en el prototipo del sistema. El esquema propuesto para etiquetar una colección de imágenes se presenta en la Figura 25.

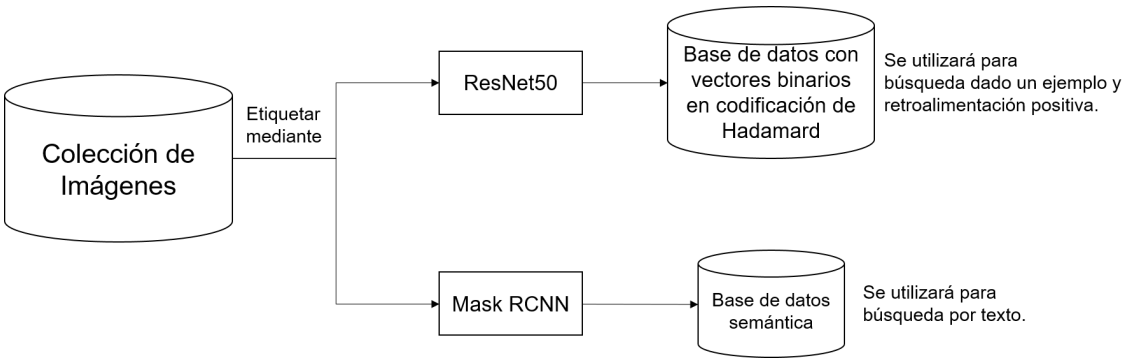


Figura 25. Esquema propuesto para realizar etiquetado de imágenes de una colección de imágenes.

4.1.1. Códigos de Hadamard

El código de Hadamard (Reed y Solomon, 1960) es un código de corrección de errores que se utiliza para la detección y corrección de errores cuando se transmiten mensajes por canales muy ruidosos o poco fiables. Son un caso particular de los códigos Reed-Solomon (Wicker y Bhargava, 1999), corresponden al primero código Reed-Solomon. La idea principal de estos códigos es incrementar la distancia de Hamming entre palabras. Una palabra es una cadena finita de bits que son manejados como un conjunto por una computadora.

Las redes neuronales utilizan el etiquetado de una clase (del inglés *One-hot labeling*). Este etiquetado funciona de la siguiente manera: si tenemos una red para clasificar imágenes en 10 clases diferentes, como salida obtendremos un grupo con 10 de bits entre los cuales el bit (1) representará la clase a la cual fue asignada la imagen y las otras 9 clases contendrán un (0), ya que la imagen no fue clasificada en esas clases. El etiquetado de una clase tiene vulnerabilidad a ataques adversarios ya que solo cambiando un bit puede producir un resultado diferentes. En (Hoyos *et al.*, 2021) se propone el uso de códigos de Hadamard como etiquetado de clase para entrenar modelos de redes neuronales como sistema defensivo contra los ataques adversarios.

Para construir los códigos de Hadamard en un problema de clasificación se usa la siguiente regla: Supongamos que necesitamos códigos de longitud 2^k, en donde hay como máximo 2^k clases posibles, es decir, en caso de querer entrenar una red con el conjunto de datos de ImageNet que cuenta con 1,000 clases, tendremos un valor de k=10, obteniendo una longitud de 1,024. Una de las ventajas de utilizar estos códigos

de Hadamard es que la distancia de Hamming entre vectores de diferentes clases es $m/2$, siendo m el número de clases. Esto quiere decir, que los vectores que pertenecen a una misma clase tendrán una distancia de Hamming pequeña. Mientras que vectores de clases diferentes tendrán una distancia de Hamming grande.

En la Figura 26 se muestra un ejemplo de etiquetado y codificación Hadamard en la base de datos de imágenes de dígitos manuscritos (MNIST) (Deng, 2012). Se puede ver que el conjunto de datos MNIST se divide en 10 clases, por lo cual, el vector en codificación de Hadamard tendrá una longitud de 16 bits.

Clase	Etiquetado de una clase	Codificación de Hadamard
0	1 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1	0 1 0 0 0 0 0 0 0 0	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
2	0 0 1 0 0 0 0 0 0 0	1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
3	0 0 0 1 0 0 0 0 0 0	1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1
4	0 0 0 0 1 0 0 0 0 0	1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
5	0 0 0 0 0 1 0 0 0 0	1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1
6	0 0 0 0 0 0 1 0 0 0	1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1
7	0 0 0 0 0 0 0 1 0 0	1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
8	0 0 0 0 0 0 0 0 1 0	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
9	0 0 0 0 0 0 0 0 0 1	1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1

Figura 26. Ejemplo de etiquetado de una clase y codificación de Hadamard para las clases del conjunto de datos MNIST (Hoyos *et al.*, 2021).

Para obtener los vectores binarios en codificación de Hadamard (que desde ahora también se les nombrará como «la codificación de Hadamard») al procesar una imagen en una red neuronal profunda, se realizaron algunos cambios a la arquitectura del modelo a utilizar. Primero, se debe cambiar el hiperparámetro de número de clases, por lo general, este valor corresponde al número de clases de nuestro conjunto de datos, sin embargo, ahora se asignará el valor de la longitud del código de Hadamard. Por ejemplo, si tenemos un conjunto de datos con 1,000 clases, el código de Hadamard tendrá una longitud de 1,024, por la regla de 2^k. Al igual que en la obtención de las características profundas, se debe eliminar o remover la última capa que por lo general, corresponde a la etapa de clasificación. Ahora, la salida de nuestro modelo será un vector de números reales con dimensión igual al número de clases asignado. Este vec-

tor será decodificado como en un canal de transmisión de mensajes. Para decodificar el vector y convertir a codificación de Hadamard, se deberá agregar una capa al final del modelo con la función de activación *HardTanh*. Esta función se encuentra definida por la Ecuación 1 y se aplica a cada elemento del vector. Sin embargo, el vector que obtendremos como salida de nuestro modelo tendrá valores de -1 y 1. Para convertirlo a binario, se deberá recorrer el vector y cambiar por 0, todo aquel valor que sea menor o igual a 0. Con los cambios anteriores, cada vez que se procese una imagen en nuestro modelo, obtendremos un vector binario en codificación de Hadamard.

$$HardTanh(x) = \begin{cases} 1 & \text{si } x > 1 \\ -1 & \text{si } x < 0 \\ x & \text{de lo contrario} \end{cases} \tag{1}$$

En general, los cambios que se deben realizar en un modelo para obtener vectores binarios en codificación de Hadamard son dos, los cuales son: (1) reemplazar el número de clases correspondiente con el valor resultante al calcular 2^k, y (2) agregar una capa con la función de activación *HardTanh*. Además, fuera del modelo, será necesario recorrer el vector obtenido como salida del modelo para cambiar los valores menores a cero por ceros. Así obtendremos vectores binarios en codificación de Hadamard de una imagen.

Lo realizado en está sección se le conoce como encaje de imágenes. Un encaje de imágenes se refiere a un conjunto de técnicas utilizadas para reducir la dimensionalidad de los datos de entrada procesados por redes neuronales, incluidas las redes neuronales profundas. Es importante para la clasificación de imágenes, ya que las imágenes tienen una gran dimensión. Por ejemplo, una imagen de 20 megapíxeles con 3 capas RGB representa tener 60 millones de valores enteros como información total almacenada en la imagen. Con los códigos de Hadamard, se construyó un encaje binario que representa a la imagen en cadenas de bits.

4.1.2. Etiquetado semántico

Se etiquetó una colección de imágenes mediante el uso de un modelo de segmentación panóptica. El modelo seleccionado fue Mask RCNN. Este modelo permite detectar

de una imagen 80 diferentes objetos y 54 diferentes cosas, obteniendo un vector de 134 dimensiones.

El etiquetado semántico de imágenes únicamente se realizó para construir un módulo del sistema de recuperación de imágenes que permita realizar búsquedas por texto. Por lo tanto, no se realizó un análisis a profundidad de los diferentes modelos que permiten realizar etiquetado semántico.

La razón por la que se incluyó el etiquetado semántico es debido al análisis presentado en (Schoeffmann *et al.*, 2021). En este análisis se menciona que la búsqueda por texto es una herramienta esencial en los sistemas de recuperación de información, ya que el usuario se encuentra acostumbrado a interactuar mediante consultas de texto. Esta conclusión se toma con base en los 10 años que tiene la competencia *Video Browser Showdown*.

Cabe aclarar que en esta tesis no fue incluido un análisis del etiquetado semántico de las imágenes de una colección, ya que es una tarea que se encuentra fuera de nuestro alcance. Simplemente se realizó el etiquetado semántico para construir el módulo de búsqueda por texto y demostrar cómo funcionaría la búsqueda por texto en nuestro prototipo de un sistema de recuperación de imágenes.

4.2. Recuperación de imágenes

El algoritmo *kNN* es ampliamente utilizado en tareas de recuperación de información, ya que permite recuperar un número k de objetos semejantes. También puede ser utilizado en sistemas de recuperación de imágenes, ya que las características profundas tienen la propiedad de obtener elementos semejantes con el cálculo de la distancia entre vectores. Sin embargo, elegir el valor de k es una tarea bastante compleja. Además, en sistemas que pueden llevarse a producción, esto no resulta una buena práctica. Es por ello que se propone el uso del grafo HSP (Chávez *et al.*, 2005) para recuperar imágenes de una colección. El algoritmo HSP permite recuperar imágenes semejantes sin la necesidad de definir parámetros, evitando el problema del algoritmo *kNN* al tener que seleccionar el valor de k. Además, la vecindad de imágenes tiene la propiedad de ser diversa, es decir, contiene imágenes de una misma clase que cuentan con diferentes atributos.

4.2.1. Algoritmo HSP

En (Talamantes y Chávez, 2021) se utiliza el grafo HSP (Véase sección 2.6) para diseñar un algoritmo que permita realizar tareas de clasificación. Este algoritmo mantiene la simplicidad de *kNN*. Además, tiene la propiedad de seleccionar vecinos similares y diversos a la vez, lo que da una vecindad representativa para cada consulta. Cada consulta es capaz de seleccionar sus vecinos, por lo cual, no es necesario definir hiperparámetros.

El algoritmo HSP, al igual que el algoritmo *kNN*, calcula la distancia entre una consulta y los objetos de una base de datos. Esto representa la construcción del grafo en tiempo lineal. Para solucionar este problema, en (Talamantes y Chávez, 2021), se propone una alternativa para calcular el HSP dentro de una circunferencia de cierto radio.

Un radio natural son los k vecinos más cercanos. Calculando el HSP dentro de un área definida por los vecinos más cercanos, las consultas tienen la probabilidad de acelerarse con el uso de un índice. Además, si se usa un índice probabilístico para calcular los *k* vecinos más cercanos de la consulta, llamamos a la vecindad resultante HSP asintótico probabilístico. Por lo cual, se utilizó la versión del algoritmo HSP llamada «HSP asintótico probabilístico».

El índice probabilístico seleccionado es el HNSW, descrito en la sección 2.5.1. La recuperación de imágenes semejantes se realiza obteniendo los *k* vecinos más cercanos y después se construye el grafo HSP de una consulta con los vecinos más cercanos obtenidos.

El algoritmo HSP hace un muestreo del espacio dando variedad. En cada iteración, después de elegir un vecino, se agregan a un espacio prohibido aquellos objetos que se encuentran más cercanos a ese vecino que a la propia consulta; estos nodos ya no necesitan estar en la vecindad de la consulta porque ya existe un nodo que los represente. Por lo tanto, el hecho de que se defina el espacio prohibido es equivalente a que se nombre un representante por dirección. Con este procedimiento se elimina la redundancia de la vecindad y se obtienen vecinos que son diferentes entre ellos, pero a su vez, cercanos a la consulta.

4.3. Desarrollo de un sistema de recuperación de imágenes

El desarrollo de un sistema de imágenes se comprende de tres fases: (1) extracción de características, (2) desarrollo del *backend*, y (3) desarrollo del *frontend*. La etapa de extracción de características fue explicada en la sección 4.1, en donde se utilizaron los códigos de Hadamard para obtener una base de datos de vectores binarios en codificación de Hadamard y la segmentación semántica para obtener una base de datos semántica. Se utilizó el conjunto de datos Places365 (Zhou *et al.*, 2018) como base de datos de nuestro sistema. En la Figura 27 se muestra la arquitectura del sistema construido.

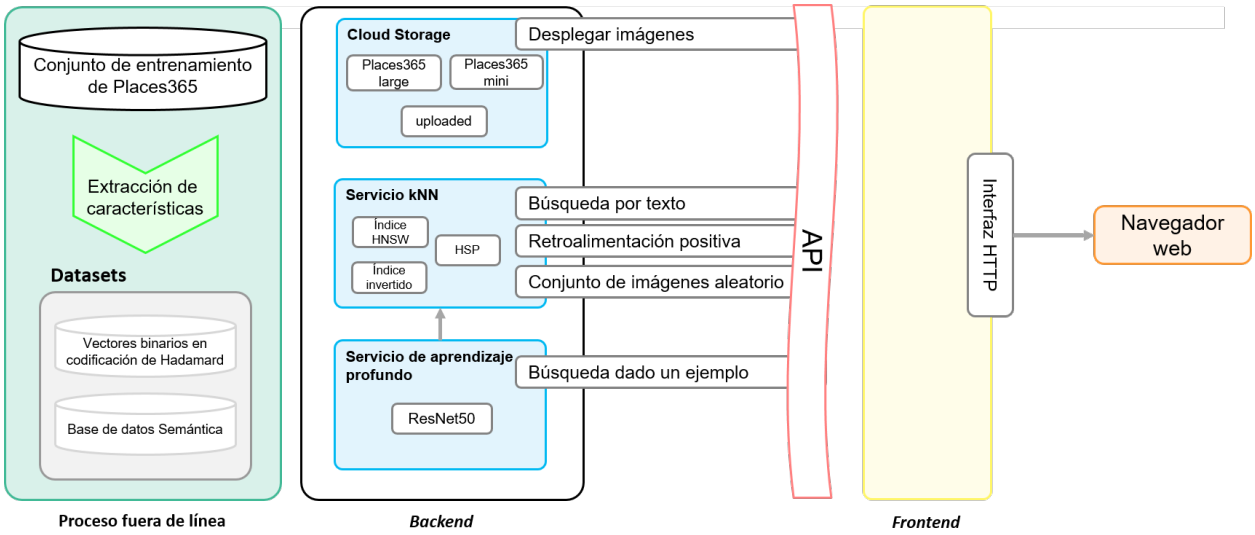


Figura 27. Arquitectura del sistema de recuperación de imágenes desarrollado.

4.3.1. *Backend*

El *backend* fue construido utilizado el framework Flask (Para más detalle véase la sección 3.2.2). La arquitectura fue diseñada utilizado un enfoque de microservicios (Nadareishvili *et al.*, 2016). Este enfoque permite a las aplicaciones escalar de una manera más sencilla.

4.3.1.1. Indexamiento de datos

Lo primero que se realizó fue indexar los datos de la colección de imágenes Places365. Para ello se utilizó la implementación *hnswlib*[1] del índice probabilístico HNSW. Este índice permite acelerar las consultas del vecino más cercano. Para construir el índice se indexaron todos los vectores binarios en codificación de Hadamard obtenidos de ResNet50 al procesar todas las imágenes del conjunto de entrenamiento de Places365.

4.3.1.2. Índice invertido

Un índice invertido es una estructura de datos de índice que almacena un mapeo de contenido, como palabras o números, a sus ubicaciones en un documento o conjunto de documentos. En la Figura 28 se muestra un ejemplo de un índice invertido.

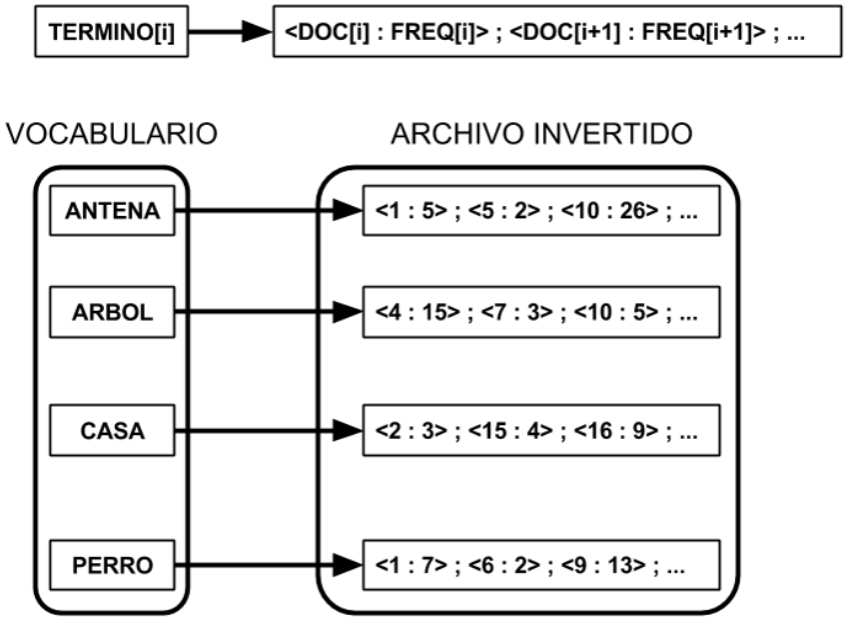


Figura 28. Se muestra un ejemplo de índice invertido.

Para construir el índice se utilizó la base de datos semántica obtenida al procesar el conjunto de entrenamiento de Places365 en el modelo de segmentación panóptica Mask RCNN. Los vectores semánticos de 134 dimensiones fueron utilizados para

[1] Hnswlib - fast approximate nearest neighbor search (2021). Recuperado el 3 de mayo de 2021 de: https://github.com/nmslib/hnswlib

construir una matriz de TF-IDF.

La frecuencia del término (TF, del inglés *Term Frequency*) determina la frecuencia relativa de un término específico, una palabra o una combinación de palabras, en un documento; un nuestro caso un vector. Mientras que la frecuencia inversa del documento (IDF, del inglés *Inverse Document Frequency*), completa el análisis de evaluación de los términos y actúa como corrector del TF. La Frecuencia Inversa de Documento es muy importante ya que incluye el cálculo de la frecuencia de documento de términos específicos: compara el número de todos los documentos disponibles con el número de documentos que contienen el término. Y para finalizar, el logaritmo se encarga de comprimir los resultados. La Ecuación 2 define TF-IDF.

$$tf - idf(t, D) = \frac{t \in d}{T \in D} * log\left(\frac{D}{d \in D : t \in d}\right) \tag{2}$$

donde t es la cantidad de veces que un término se encuentra en un documento d; T es la cantidad de palabras totales en el documento d; D es el número total de documentos; y el término $d \in D$: $t \in d$ es la suma de todas las veces que el término t aparece en todos los documentos.

La matriz de TF-IDF de la base de datos semántica fue utilizada para construir el índice invertido. El índice invertido contiene 134 términos que corresponden a los objetos y cosas que detecta de una imagen la red Mask RCNN. Después, cada vector de la matriz se indexó colocando un registro en el término que aparece en el vector, agregando un par que contiene el id del vector y el valor de TF-IDF. Cuando se terminó de indexar toda la matriz, cada uno de los 134 términos contiene una lista con el id de la imagen en donde se encuentra ese término y el valor TF-IDF correspondiente.

4.3.1.3. Microservicios

La arquitectura de microservicios implementada contiene tres servicios, los cuales son los siguientes:

1. **Cloud Storage**: Se encarga de almacenar imágenes del conjunto de datos de Places365. Dentro del *bucket*, llamado Places365, se crearon tres carpetas, las

cuales son:

a) **Places365 *large***: Almacena las imágenes de Places365 en tamaño original.

b) **Places365 *mini***: Almacena las imágenes de Places365 en tamaño 224 x 224.

c) ***Uploaded***: Guarda imágenes a ser procesadas para búsqueda dado un ejemplo.

2. **Servicio kNN**: Se encarga de inicializar el índice probabilístico HNSW con los datos de la base de datos de vectores binarios en codificación de Hadamard con 512 de dimensión, inicializa el índice invertido con la base de datos semántica e implementa el algoritmo basado en instancias HSP utilizado para mostrar imágenes semejantes.

3. **Servicio de aprendizaje profundo**: Se encarga de procesar imágenes con el modelo ResNet50 para realizar búsqueda dado un ejemplo. Este modelo tiene las modificaciones en la arquitectura para obtener vectores binarios en codificación de Hadamard al procesar una imagen.

Obtenemos muchas ventajas al dividir la aplicación en microservicios. La primera ventaja es que al utilizar Cloud Storage nos permite acceder a objetos de baja latencia, almacenamiento de bajo costo, encriptación de datos, y administración de identidades y acceso. La segunda ventaja es dividir la carga de trabajo del uso de una Unidad de Procesamiento de Gráficos (GPU, del inglés *Graphics Processing Unit*) en un solo servicio. Adquirir una GPU en un proveedor de la nube, por ejemplo Google, tiene un alto costo. En caso de adquirir una GPU en Google Cloud se podría inhabilitar el uso de la GPU para no gastar recursos y reducir costos.

4.3.2. *Frontend*

La interfaz de usuario fue construida utilizando la biblioteca de JavaScript React (Para más detalle véase la sección 3.2.1). El diseño de la interfaz se encuentra inspirado en la interfaz usada en otros sistemas de recuperación que utilizan aprendizaje interactivo. La estructura de la interfaz se encuentra compuesta por cinco componentes que permiten: (1) Generar un conjunto aleatorio de imágenes, (2) Realizar búsqueda

por texto, (3) Hacer una búsqueda dado un ejemplo, (4) Sección para realizar retroalimentación positiva, y (5) Una rejilla de 6 x 4 que muestra hasta 24 imágenes, se utiliza para mostrar los resultados al realizar búsquedas o generar un conjunto aleatorio de imágenes. Además, se incluyó una sección para mostrar el historial de búsqueda, en donde se muestra en forma de pila (la última imagen que se utilizó para buscar es la primera en mostrar, mientras que la primera imagen que se uso para buscar se muestra al final). Esta sección no tiene ninguna funcionalidad, simplemente es para visualizar un historial de búsqueda.

4.3.2.1. Retroalimentación positiva

En el sistema se pueden realizar búsquedas por texto o dado un ejemplo, sin embargo, la herramienta principal para navegar y recuperar imágenes es la retroalimentación positiva.

La retroalimentación positiva es una variante del algoritmo de aprendizaje interactivo que solo utiliza retroalimentación positiva, es decir, únicamente se etiquetan imágenes como relevantes o positivas. La retroalimentación positiva para navegar en una colección multimedia funciona de la siguiente manera:

1. Mediante un conjunto aleatorio de imágenes, el usuario selecciona imágenes semejantes a la imagen objetivo (marcándolas como relevantes o positivas) y con la imagen o imágenes seleccionadas se realiza una búsqueda; esto se conoce como ciclo o ronda de retroalimentación. Este ciclo se repite hasta que el usuario esté satisfecho.
2. Para consultas por texto, el usuario ingresa una consulta y se muestran imágenes similares a la consulta, obteniendo un conjunto de imágenes relevantes y se procede a seguir el flujo del inciso (a).

La consulta por texto no utiliza un conjunto aleatorio de imágenes, sino que este es sustituido con el conjunto de imágenes obtenido de la búsqueda realizada.

4.4. Evaluación

Las características profundas (Véase sección 2.2.2) son ampliamente utilizadas en tareas de recuperación y permiten obtener buenos resultados. Dentro del área de sistemas de recuperación es fundamental que las características que se utilicen como objetos tengan un alto grado de pureza. Es decir, si se quiere buscar un perro con una búsqueda dado un ejemplo, el sistema debe regresar imágenes de perros o que contengan perros.

Sabemos que la representación con características profundas es una buena opción para tareas de clasificación, recuperación y navegación, sin embargo, nuestra hipótesis es que la representación con codificación de Hadamard (Véase sección 4.1.1) es tan buena o mejor que la representación con características profundas para tareas de recuperación. En la Figura 29 podemos ver un caso específico en el que la codificación de Hadamard obtiene mejores resultados que las características profundas. En el ejemplo se quiere clasificar una consulta (o imagen) del conjunto de validación de Places365, con la etiqueta *airfield* como *ground truth*. Para clasificar la consulta se construyó un clasificador kNN con un valor de $k=10$ para codificación de Hadamard y características profundas. Los resultados de los diez vecinos más cercanos muestran que con el uso de codificación de Hadamard se obtienen ocho imágenes con la misma etiqueta que la consulta, mientras que con características profundas únicamente seis vecinos. Por lo tanto, para este ejemplo en específico la codificación de Hadamard presenta mejores candidatos. Sin embargo, cabe mencionar que tanto en características profundas como en codificación de Hadamard los vecinos más cercanos con una etiqueta diferente a la consulta, son muy semejantes a la consulta. Además, en ambos casos estos vecinos contienen la etiqueta *runway* (pista, en español) que es muy similar a *airfield* (aeródromo, en español).

Para probar nuestra hipótesis se tomaron los conjuntos de datos de «ImageNet» y «Places365» con sus respectivos conjuntos de entrenamiento y validación. La Tabla 2 muestra las características principales de cada conjunto de datos o *datasets*. Los conjuntos fueron etiquetados utilizando las técnicas de características profundas y codificación de Hadamard mediante el uso de un modelo ResNet50.

La idea de los experimentos es utilizar cada imagen del conjunto de validación de

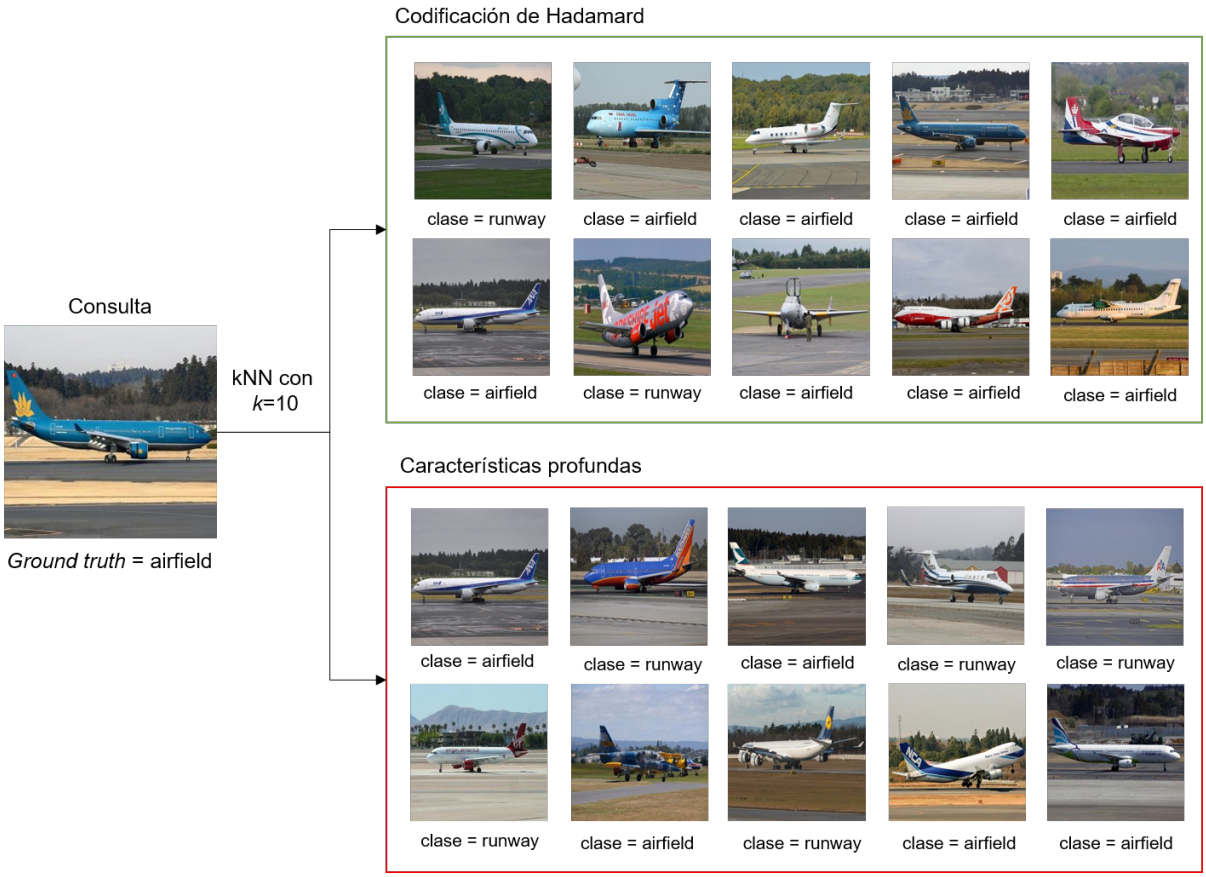


Figura 29. Caso específico en donde la codificación de Hadamard obtiene mejores resultados que características profundas. Al clasificar la consulta mediante kNN con una K con valor de 10, obtenemos con codificación de Hadamard ocho vecinos de la misma clase que la consulta, mientras que con características profundas obtenemos solamente seis vecinos .

Places365 e ImageNet y realizar consultas a su respectiva base de datos (conjunto de entrenamiento). Para realizar las consultas se utilizaron los algoritmos kNN y el algoritmo HSP asintótico probabilístico. Estos algoritmos fueron seleccionados dada su simplicidad y además, permiten calcular la distancia entre vectores. Por lo tanto, se realizaron consultas utilizando características profundas y codificación de Hadamard utilizando los algoritmos HSP y kNN con los conjuntos de datos Places365 e ImageNet.

Después de realizar la consulta, se calculó la pureza de la vecindad resultante. Esta vecindad se conforma de vecinos relevantes de la consulta. Un vecino relevante se define como un objeto que es semejante a la consulta. Por ejemplo, al utilizar el algoritmo kNN, la cantidad de vecinos relevantes a recuperar se define con el valor del hiperparámetro k. Mientras que con HSP, la cantidad de vecinos relevantes a recuperar se desconoce hasta que se construye el grafo HSP.

Tabla 2. Características de los conjuntos utilizados en los experimentos (Zhou *et al.*, 2018).

Dataset	# imágenes en CE	# imágenes en CV	# clases	Resolución	Etiqueta de clase
ImageNet	1,281,167	50,000	1,000	≈ 500×400	Objetos
Places365	1,803,460	36,000	365	≥ 200×200	Escena de interior/exterior

CE: Conjunto de entrenamiento; CV: Conjunto de validación

Para calcular la pureza, se utilizó la Ecuación 3, en donde se mide el porcentaje de vecinos relevantes que tienen la misma clase de la consulta. Esta métrica es similar a la que se utiliza en aprendizaje no supervisado para el análisis de clústers. La pureza tiene un rango entre 0 y 1. Dado que el calculo de la pureza se realiza por consulta, se calculó el promedio de la pureza por clase.

$$Pureza(q) = \frac{\text{vecinos relevantes con la misma etiqueta que } q}{\text{número total de vecinos relevantes}} \tag{3}$$

El algoritmo HSP se construye sobre un radio, es decir, una cantidad k de vecinos más cercanos. Por lo tanto, para ver el comportamiento en radios de diferentes tamaño, se definieron los valores de k= 25, 50, 75, 100, 150 y 200. Considerando que al utilizar el algoritmo HSP no conocemos la cantidad de vecinos resultantes y varía con respecto a la consulta, el resultado esperado es que al aumentar el tamaño de los vecinos más cercanos, la cantidad de vecinos relevantes también aumente. Por otra parte, en kNN es necesario definir un valor de k, se decidió de forma empírica que el valor de k siempre sea 10. Por lo tanto, la cantidad total de vecinos relevantes con kNN siempre será igual a 10, mientras que con HSP la cantidad será diferentes para cada consulta.

Con lo anterior, se propone un conjunto de pruebas para comparar el uso de características profundas y la codificación de Hadamard. El primer experimento es comparar las características profundas contra la codificación de Hadamard utilizando los algoritmos kNN y HSP en los conjuntos de datos Places365 e ImageNet. Para compararlos, se tomará el valor de la pureza de cada clase y se calculará el cociente, es decir, dividir el valor de pureza de clase en codificación de Hadamard entre el valor de pureza de clase en características profundas. Si el valor resultante es mayor a 1, quiere decir que en esa clase los vectores binarios en codificación de Hadamard son mejores. Si el valor es igual a 1, quiere decir que son igual de eficientes en esa clase. Por último, si el

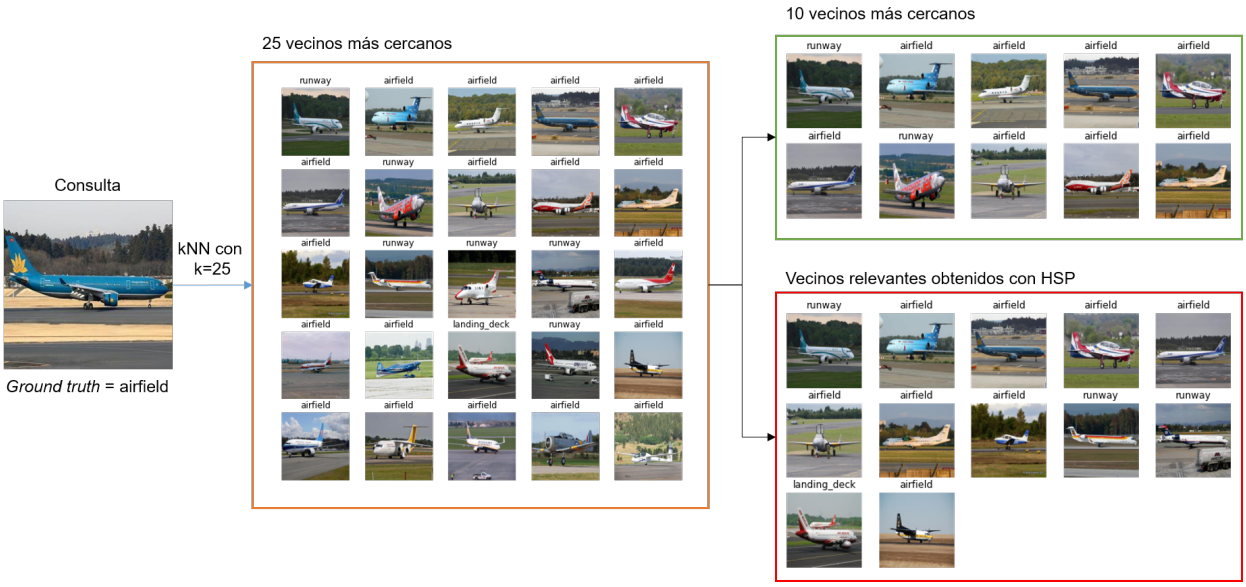


Figura 30. Caso específico en donde el algoritmo HSP tiene una mayor pureza que kNN.

valor es menor a 1, entonces las características profundas son mejores que los vectores binarios en codificación de Hadamard en esa clase. Este experimento se realizará tanto para kNN como para HSP en ambos conjuntos de datos.

El experimento anterior permitirá comparar la codificación de Hadamard y las características profundas en cada algoritmo. Sin embargo, en la Figura 30 se muestra un caso específico en donde el algoritmo HSP tiene una mayor pureza que kNN utilizando la codificación de Hadamard. Por lo cual, el siguiente experimento es comparar la codificación de Hadamard con los algoritmos kNN y HSP. Por lo cual se calculará el cociente entre el promedio de la pureza por clase de la codificación de Hadamard en HSP, entre el promedio de la pureza por clase de la codificación de Hadamard en kNN. En caso que el resultado sea mayor a 1, el algoritmo HSP es mejor. Mientras que si es menor a 1, el algoritmo kNN es mejor. Por último, si el resultado es igual a 1, tanto el algoritmo kNN como el algorimto HSP son igual de eficientes.

Por último, es importante realizar un experimento que muestre los vecinos relevantes recuperados por el algoritmo HSP tanto en características profundas y la codificación de Hadamard. Por lo cual, se mostrará el número de vecinos relevantes por consulta en los diferentes radios seleccionados.

En resumen, se realizarán tres experimentos: (1) Comparar la codificación de Hada-

mard contra características profundas utilizando los algoritmos kNN y HSP, (2) Comparar el algoritmo HSP contra el algoritmo kNN usando codificación de Hadamard, y (3) presentar los vecinos relevantes de HSP en los diferentes tamaños de vecinos más cercanos. Estos experimentos se realizarán utilizando el conjunto de datos de Places365 e ImageNet.

4.4.1. Recursos utilizados

Los experimentos se realizaron utilizando el servicio de *Google Colab* [2] en su versión Pro. El entorno de Colab se encuentra basado en *Jupyter Notebook* (Kluyver *et al.*, 2016). *Jupyter* es el acrónimo de Julia, Python, Tex y R, lenguajes los cuales puede ejecutar, aunque hoy en día soporta más lenguajes. *Jupyter Notebook* es un entorno informático interactivo basado en la web para crear documentos.

Las máquinas a las que se tienen acceso con *Google Colab* cuentan con 13 GB de RAM y 60 GB de almacenamiento en disco duro. Las GPU disponibles incluyen tarjetas gráficas Nvidia K80, T4, P4 y P100. En la versión Pro, es posible configurar el entorno de ejecución para una alta capacidad de RAM. El aumento en los recursos asignados por Colab en la versión Pro es considerable, teniendo acceso a máquinas de hasta 35 GB de RAM y 180 GB de almacenamiento.

Se utilizó la versión de Python predeterminada que ofrece *Google Colab*, que es Python 3.7.10. El proceso de *transfer learning* para etiquetar las imágenes y obtener las características profundas y los vectores binarios en codificación de Hadamard se llevó acabo usando la biblioteca de *PyTorch* (Paszke *et al.*, 2019), que se utiliza ampliamente en tareas de aprendizaje automático y se encuentra incluida en *Colab*.

Debido a que las bases de datos, en específico los conjuntos de entrenamiento, contienen más de un millón de elementos, se instaló la biblioteca *hnswlib* que permite hacer uso del índice HNSW (Malkov y Yashunin, 2020). Este índice permite acelerar las consultas mediante el uso de su versión probabilística de kNN. Además, se utilizó para utilizar la versión de HSP asintótico probabilístico.

Se utilizó la biblioteca NumPy (Harris *et al.*, 2020) para almacenar los vectores de

[2] Google Colaboratory (2021). Recuperado el 11 de mayo de 2021 de: `https://colab.research.google.com`

las consultas, es decir los datos de los conjuntos de validación en características profundas y en codificación de Hadamard. Adicionalmente, para guardar los resultados se utilizó la biblioteca Pandas (Wes McKinney, 2010) para manipulación y análisis de datos y por último, la biblioteca Matplotlib (Hunter, 2007) se utilizó para realizar los gráficos.

Capítulo 5. Resultados

5.1. Prototipo de un sistema de recuperación de imágenes

Se obtuvo como resultado un sistema de recuperación de imágenes. La interfaz gráfica del sistema cuenta con los siguientes componentes: (1) Buscador por texto, (2) Generar conjunto aleatorio de imágenes, (3) Retroalimentación positiva, (4) Búsqueda dado un ejemplo, (5) Historial de búsqueda, (6) Grid de imágenes, y (7) Visualización de imágenes en tamaño real. En la Figura 31 se muestra la interfaz de usuario del sistema de recuperación de imágenes desarrollado.

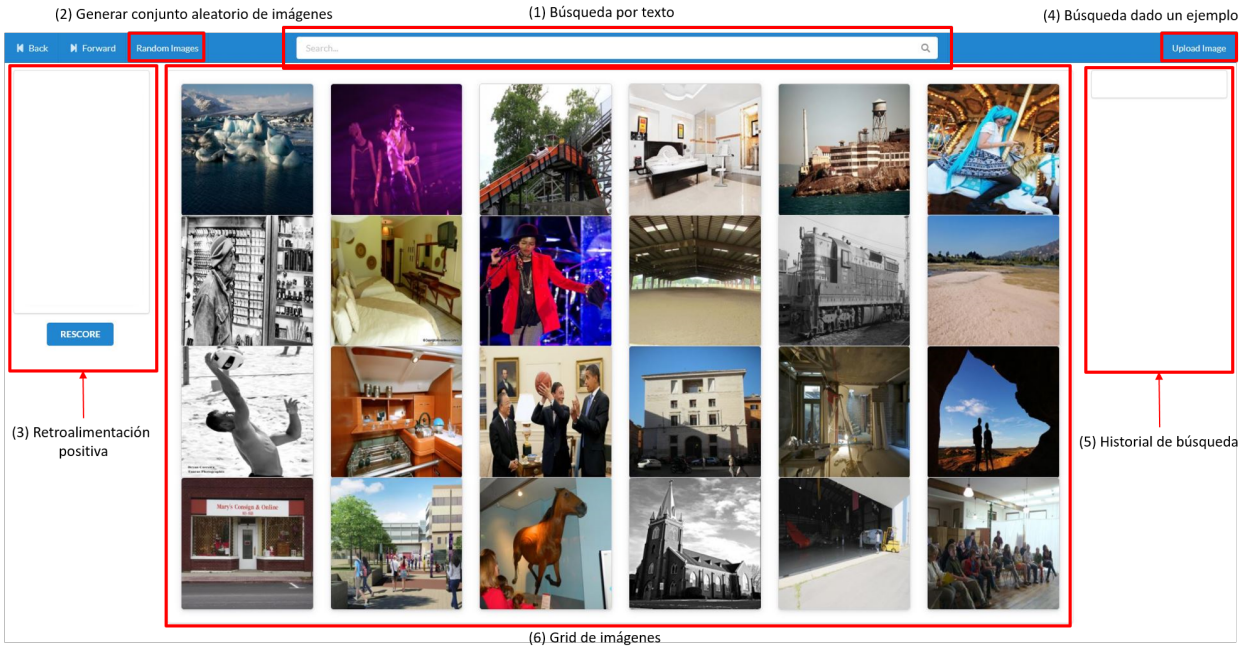
5.2. Experimentos utilizando el conjunto de datos Places365

5.2.1. Codificación de Hadamard vs. Características profundas

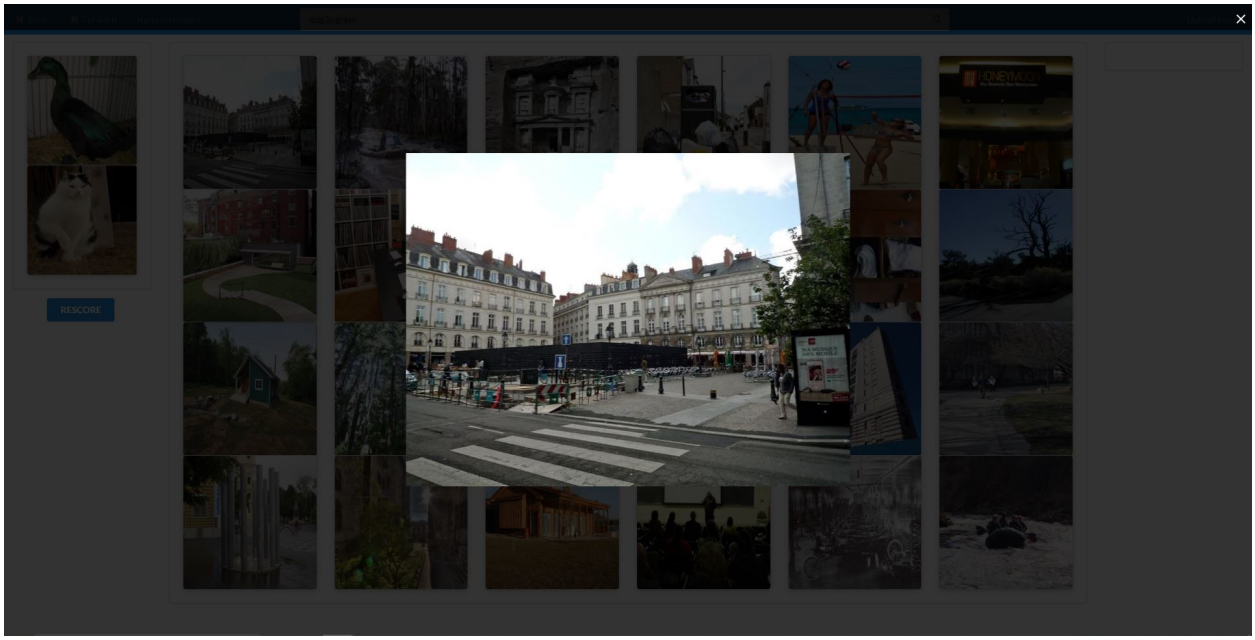
Las gráficas de la Figura 32 y 33 representan el mismo experimento, con la diferencia que se utiliza un clasificador diferente. La gráfica fue construida utilizando el cociente de las medias por clase utilizando la codificación de Hadamard y las características profundas. El objetivo del experimento es comparar la eficacia de las características profundas y la codificación de Hadamard utilizando diferentes algoritmos de aprendizaje basado en instancias. Los valores del cociente en las 365 clases, se ordenaron de forma ascendente. Si el valor del cociente es mayor a 1, significa que la codificación de Hadamard es mejor. En caso que sea menor a 1, significa que las características profundas son mejores.

En las gráficas podemos ver que tanto en los experimentos con kNN y HSP la codificación de Hadamard es mejor en más de la mitad de las 365 clases. Sin embargo, podemos ver que aproximadamente cinco clases tienen una tendencia a 0. Esto quiere decir que estas clases no son bien clasificadas al utilizar los vectores de codificación de Hadamard. Estas cinco clases son: *desert-sand*, *desert-vegetation*, *desert-road*, *field-wild* y *field-road*.

Las clases son muy similares, comparten colores y texturas semejantes; lo cual podría estar ocasionando que fueran clasificadas incorrectamente. Tanta es la similitud



(a)



(b)

Figura 31. Interfaz de usuario del sistema de navegación multimedia desarrollado. En (a) se muetran los siguientes componentes: (1) Buscador por texto, (2) Generar conjunto aleatorio de imágenes, (3) Retro-alimentación positiva, (4) Búsqueda dado un ejemplo, (5) Historial de búsqueda, y (6) Grid de imágenes. Mientras que en (b) se muestra la visualización de imágenes en tamaño real.

de las clases que son derivaciones de dos clases, es decir, que de la clase *desert* se divide en tres clases (*desert-sand*, *desert-vegetation* y *desert-road*). Mientras que la clase *field* se divide en *field-wild* y *field-road*. El conjunto de datos Places365 tiende a considerar una clase padre y de ella generar clases hijo, siendo este el caso para las cinco clases anteriores. Además, un punto a considerar, es la representación binaria de la codificación de Hadamard, lo que puede ocasionar que al calcular la distancia entre vectores, exista un empate entre distancias y dado que existen clases muy parecidas, se obtenga una clasificación incorrecta.

El comportamiento de las cinco clases ocurre para los valores de k= 25, 50, 75, 100, 150 y 200, y con ambos algoritmos (HSP y kNN).

5.2.2. El algoritmo HSP vs. el algoritmo kNN utilizando la codificación de Hadamard

En los experimentos anteriores, fue posible ver que en promedio la codificación de Hadamard tiene mejores resultados que las características profundas, tanto en el clasificador kNN, como en el clasificador HSP. Ahora bien, el siguiente experimento tiene como objetivo verificar cual de los dos clasificadores utilizados es mejor utilizando la codificación de Hadamard.

La Figura 34 presenta los resultados al comparar HSP contra kNN utilizando la codificación de Hadamard. En el eje X se encuentra el número de clases y el eje Y contiene el cociente de la pureza por clase. El cálculo del cociente se realizó dividiendo el valor de pureza de HSP entre el valor de pureza de kNN. Si el cociente es mayor a 1, significa que HSP es mejor en esa clase. Mientras que, el cociente es menor a 1, significa que kNN es mejor. En caso de obtener un 1, significa que es un empate y los dos clasificadores son igual de buenos.

Podemos ver que con los diferentes valores de k, HSP se mantiene mejor en más de la mitad de las clases. Con el aumento del valor de k, HSP amplia la diferencia, logrando ser mejor en nuevas clases. Por ejemplo, en la gráfica con una k=25 se puede ver que HSP es mejor en más de 200 clases, sin embargo, en la gráfica con una k=200 se puede ver que HSP ya es mejor en más de 250 clases. Esto ocurre porque la vecindad que toma el algoritmo HSP crece, al igual que la cantidad de vecinos

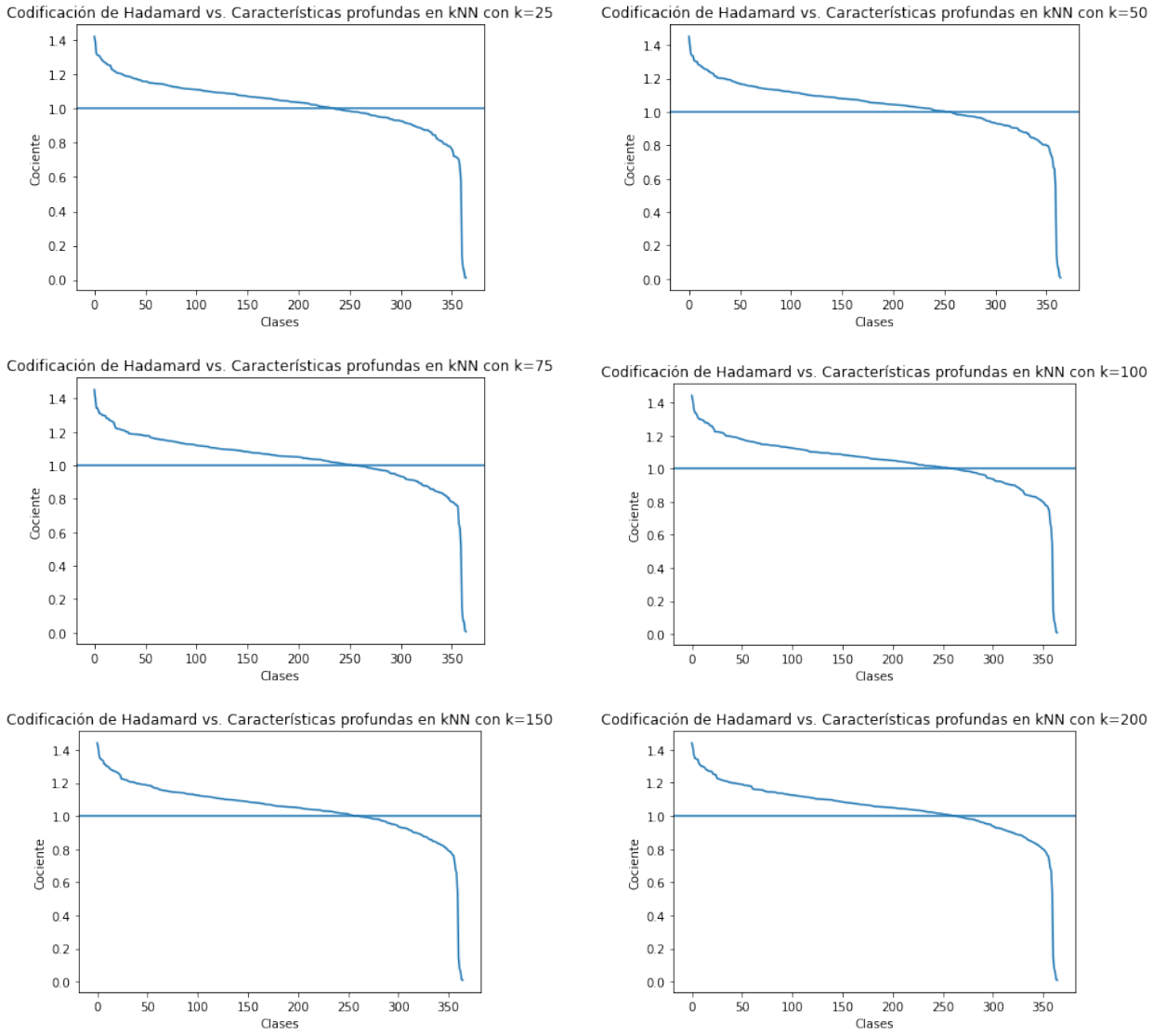


Figura 32. Comparación de la codificación de Hadamard contra las características profundas utilizando un clasificador kNN en Places365.

relevantes aumenta. Mientras que kNN se mantiene fijo con un valor de diez vecinos. El valor de diez vecinos relevantes con kNN impacta en la pureza. Se observa con base en los resultados que un valor menor de vecinos relevantes, ayudaría a obtener mejores resultados, lo cual aumentaría la pureza. Esto provocaría que kNN sea mejor que HSP en un mayor número de clases.

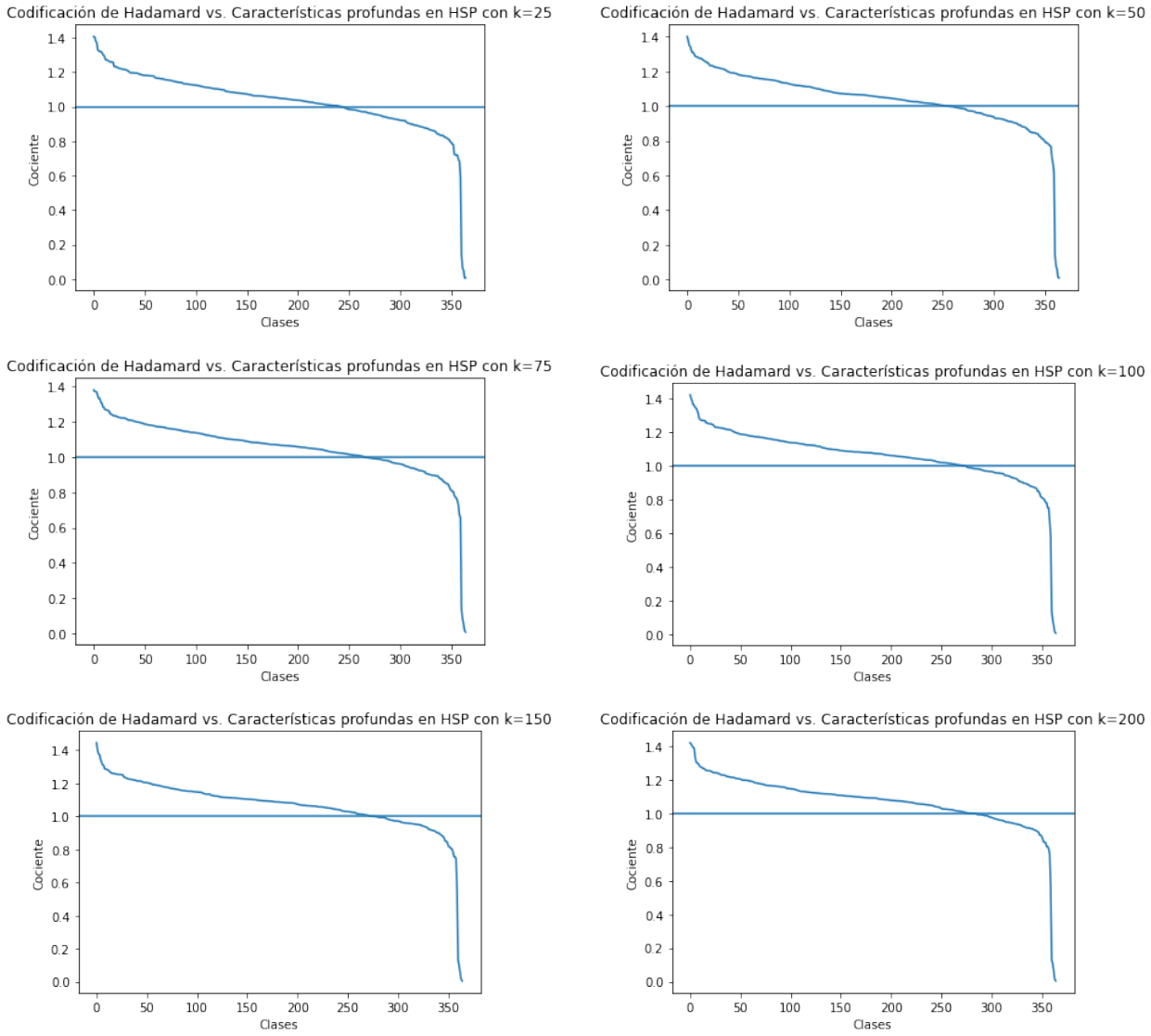


Figura 33. Comparación de la codificación de Hadamard contra las características profundas utilizando un clasificador HSP en Places365.

5.2.3. Vecinos relevantes obtenidos del algoritmo HSP

Las gráficas de la Figura 35 presentan los vecinos relevantes que devuelve el algoritmo HSP en cada consulta. La cantidad de vecinos relevantes fue ordenada de manera ascendente. En las gráficas, la línea de color naranja representa las características profundas y la línea de color azul representa la codificación de Hadamard. En el eje X tenemos el número de consultas y en el eje Y tenemos el número de vecinos relevantes.

El objetivo del experimento de la Figura 35 es ver la diversidad de vecinos relevan-

tes con cada valor de k. Podemos ver que al aumentar el valor de k, el tamaño de la vecindad también aumenta. Por ejemplo, el mayor número de vecinos relevantes con una k=25 es de 18 y cuando se aumenta el valor de k a 200, hasta 56 vecinos relevantes son obtenidos. Sin embargo, al utilizar la codificación de Hadamard se puede ver el comportamiento que muchas de las consultas devuelven un vecino relevante. Por ejemplo, con una k=25 hasta 12,000 consultas regresan un vecino, mientras que con el aumento de la k, el número de consultas con un vecino disminuye hasta 6,100. Por lo tanto, las características profundas devuelven un mayor número de vecinos relevantes que la codificación de Hadamard.

5.3. Experimentos utilizando el conjunto de datos ImageNet

Los mismos experimentos realizados con Places365 fueron ejecutados con ImageNet. Esto se realizó con el objetivo de ver el comportamiento en un conjunto de datos diferente y con mayor diversidad.

5.3.1. Codificación de Hadamard vs. Características profundas

Las Figuras 36 y 37 presentan los resultados de comparar los vectores de codificación de Hadamard contra las características profundas, tanto en kNN, como en HSP. El comportamiento del cociente por clase es diferente a lo visto en los experimentos de Places365. Podemos ver que en un inicio las características profundas son mejor por poco más de la mitad de clases. Con el aumento en el valor de k, la codificación de Hadamard comienza poco a poco a ganar terreno; volviéndose mejor en más de la mitad de clases. Por último, en la gráfica que presenta un valor de k=200, se muestra que la diferencia existente entre la codificación de Hadamard contra las características profundas es mucha mayor, ganando la codificación de Hadamard en casi 800 de las 1,000 clases de ImageNet. Este comportamiento se puede ver tanto en HSP, como en kNN. Otro punto a resaltar es la diferencia en las clases que las características profundas es mejor, siendo mejor que la codificación de Hadamard por una pequeña diferencia.

5.3.2. El algoritmo HSP vs. el algoritmo kNN utilizando la codificación de Hadamard

Con el experimento anterior, pudimos ver que la codificación de Hadamard no siempre tiene un mejor rendimiento que las características profundas. Sin embargo, con el aumento del valor de k, la codificación de Hadamard logra resultados muy buenos, logrando conseguir ser mejor en casi 800 clases de las 1,000 clases de ImageNet. En Places365 el comportamiento era diferente, ya que en todo momento la codificación de Hadamard era mejor en más de la mitad de las clases.

Ahora bien, en la Figura 38 se muestra la comparación de HSP contra kNN utilizando únicamente la codificación de Hadamard. En las gráficas se muestra que HSP siempre es mejor en más de la mitad de las clases. Además, la diferencia en las clases que kNN es mejor se ve reducida con el aumento del valor de k.

5.3.3. Vecinos relevantes obtenidos del algoritmo HSP

La Figura 39 presenta los resultados de los vecinos relevantes por consulta al construir un clasificador con HSP usando el conjunto de datos de ImageNet. En el eje X se presenta el número de consultas. El conjunto de validación de ImageNet contiene 50,000 imágenes; siendo este el número total de consultas. Mientras que el eje Y es el número de vecinos relevantes que devuelve el clasificador HSP al clasificar una consulta.

Podemos ver un fenómeno similar al que ocurre en el conjunto de datos Places365. Este fenómeno es que muchas consultas (aproximadamente 23,000) devuelven un solo vecino relevante. Esta cantidad de consultas disminuye con el aumento del valor de k. También aumenta el máximo número de vecinos relevantes. Además, podemos ver que las características profundas tienen una mejor diversidad y devuelven un mayor número de vecinos relevantes que la codificación de Hadamard.

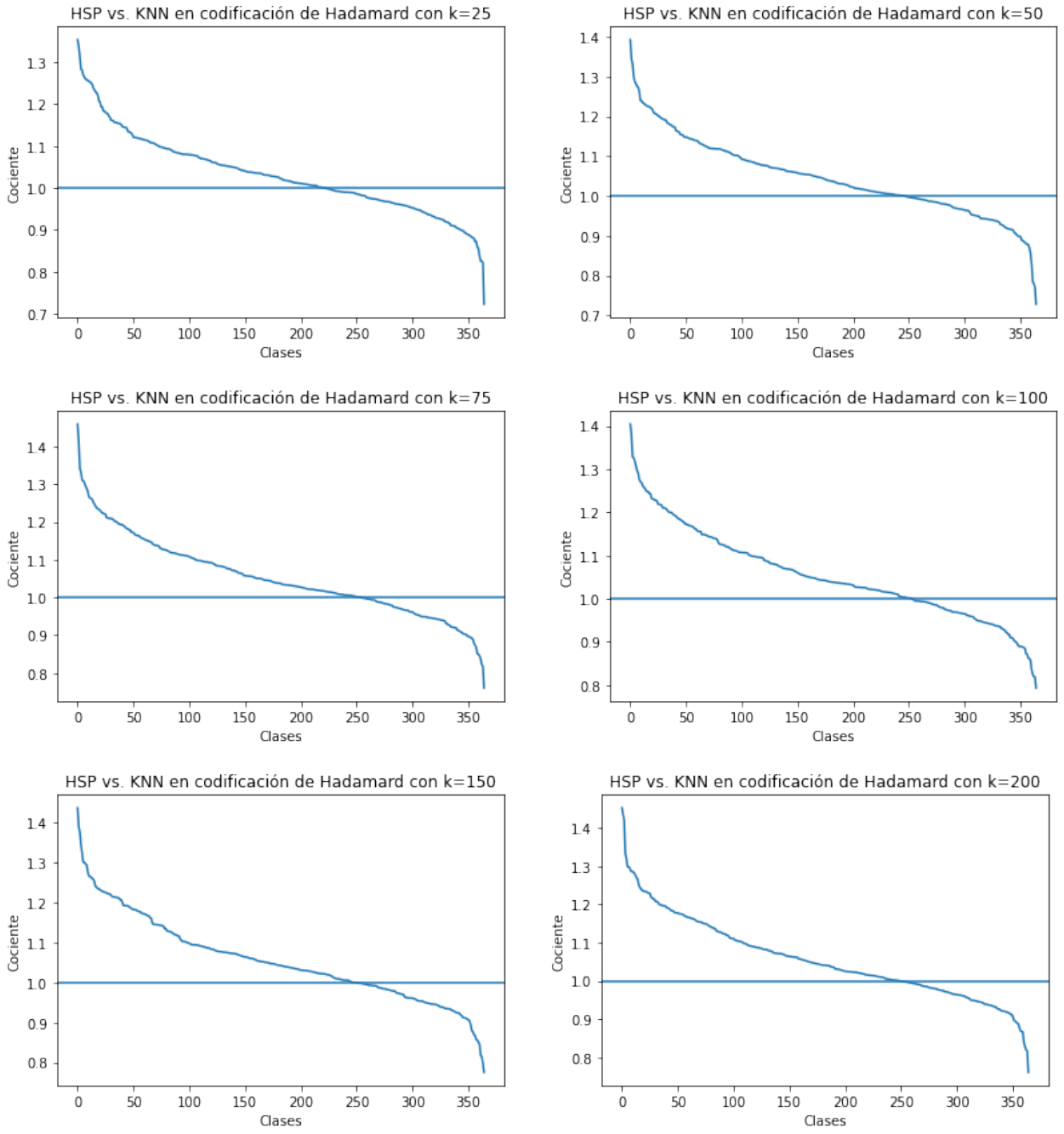


Figura 34. Comparación de HSP contra kNN utilizando codificación de Hadamard en Places365.

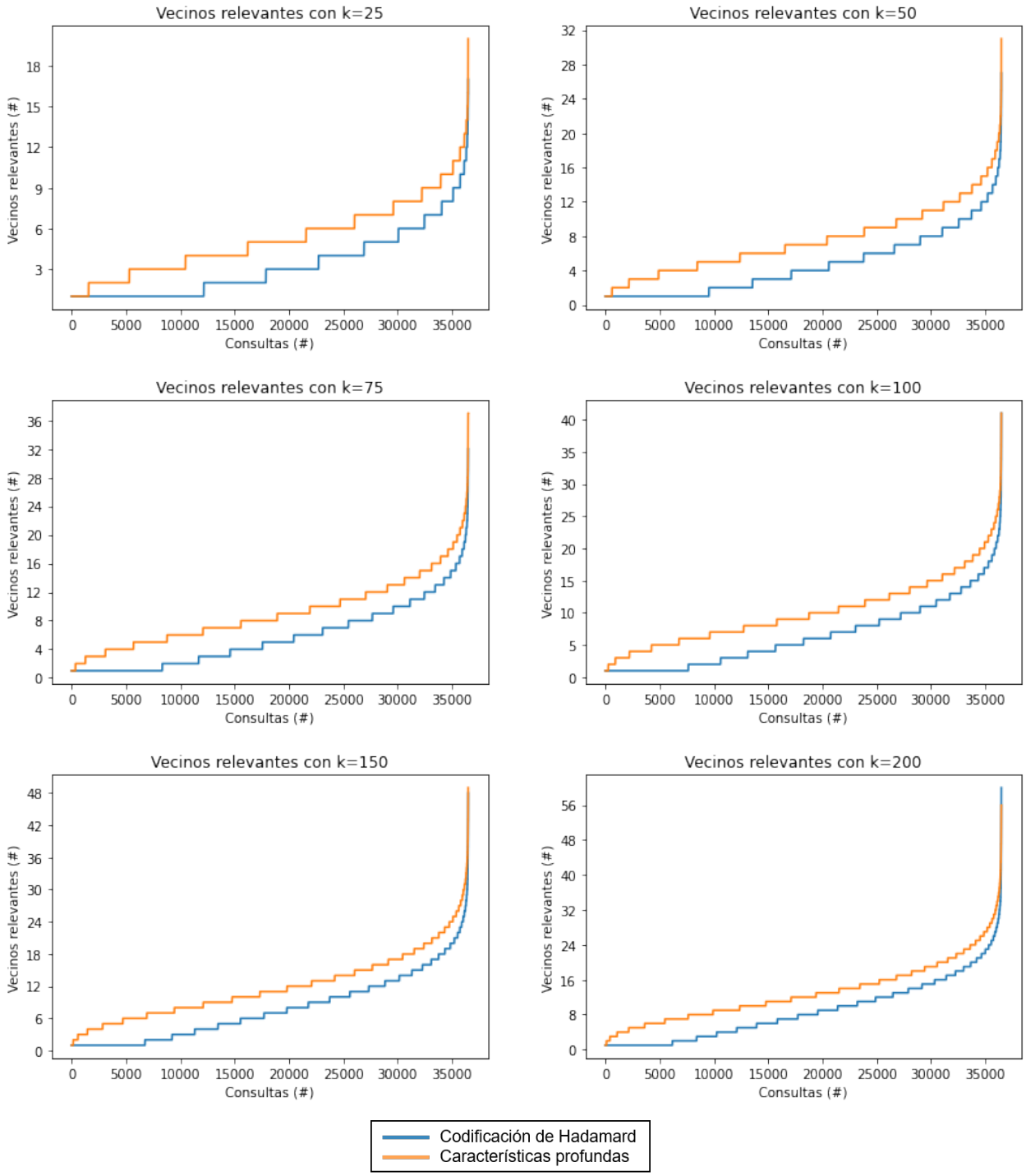


Figura 35. Comparación de vecinos más cercanos de HSP con variaciones de k en Places365. La línea naranja representa las características profundas y la línea azul representa la codificación de Hadamard.

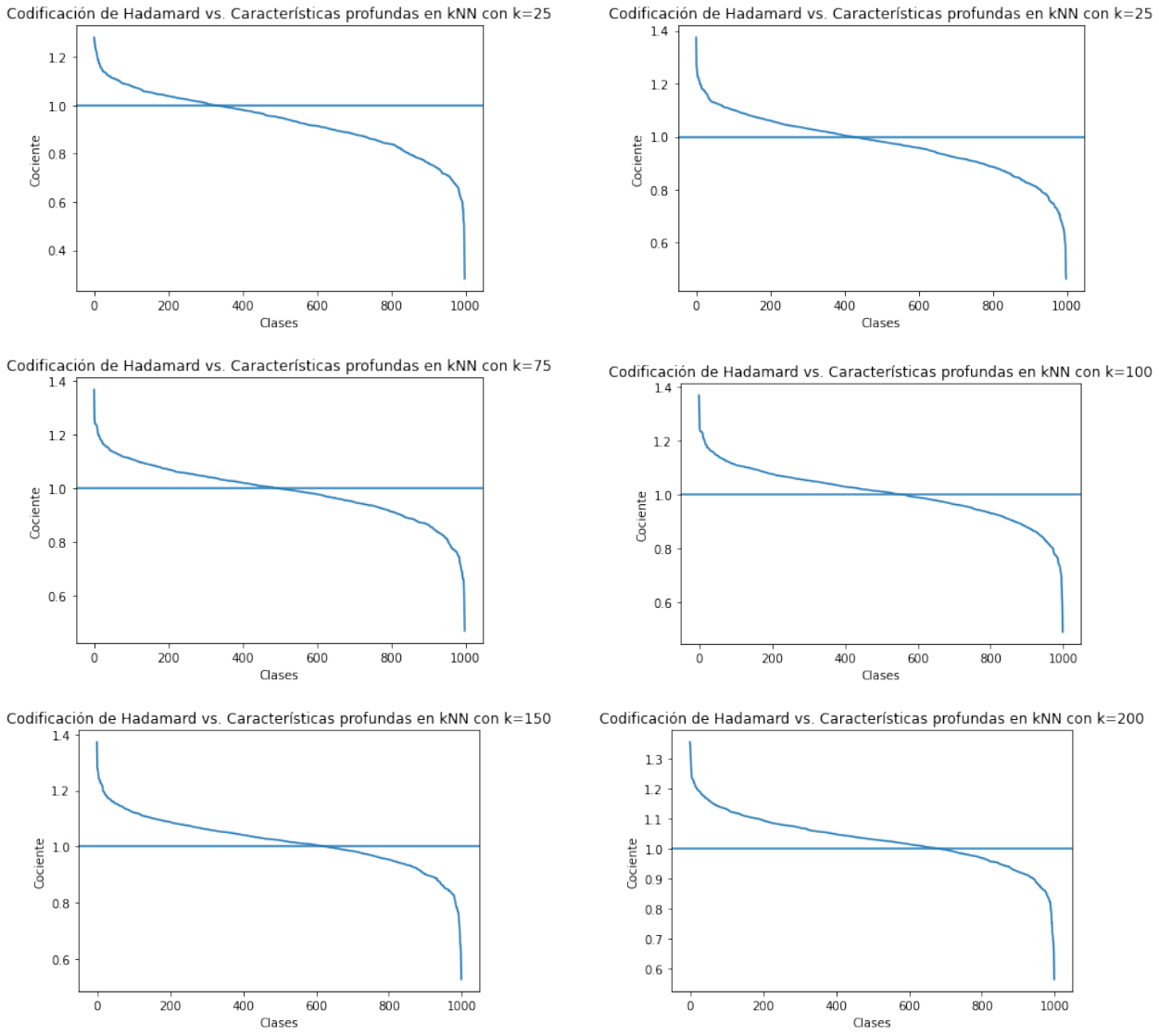


Figura 36. Comparación de la codificación de Hadamard contra las características profundas utilizando un clasificador KNN en ImageNet.

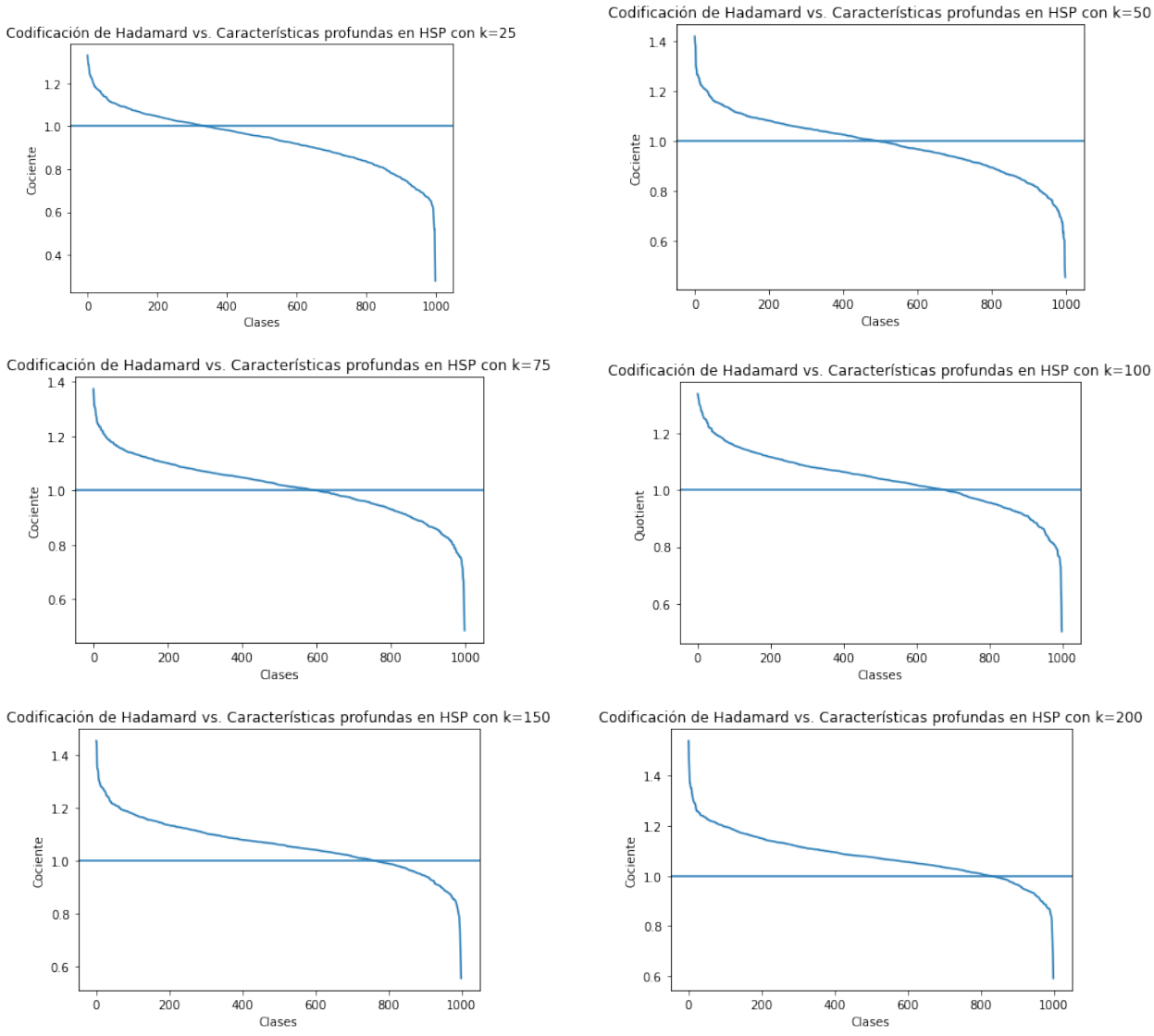


Figura 37. Comparación de codificación de la codificación de Hadamard contra las características profundas utilizando un clasificador HSP en ImageNet.

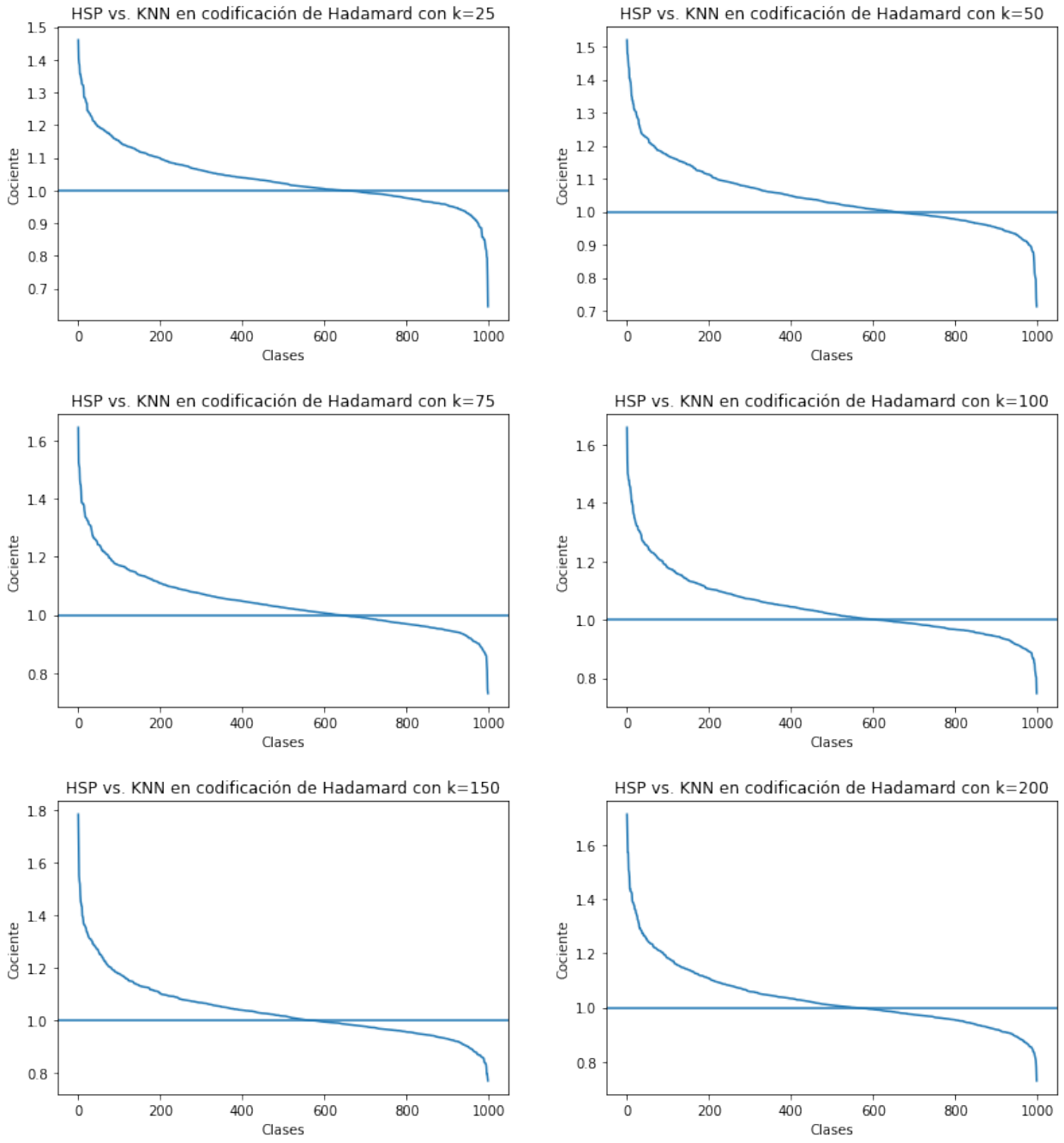


Figura 38. Comparación de HSP contra kNN utilizando la codificación de Hadamard en ImageNet.

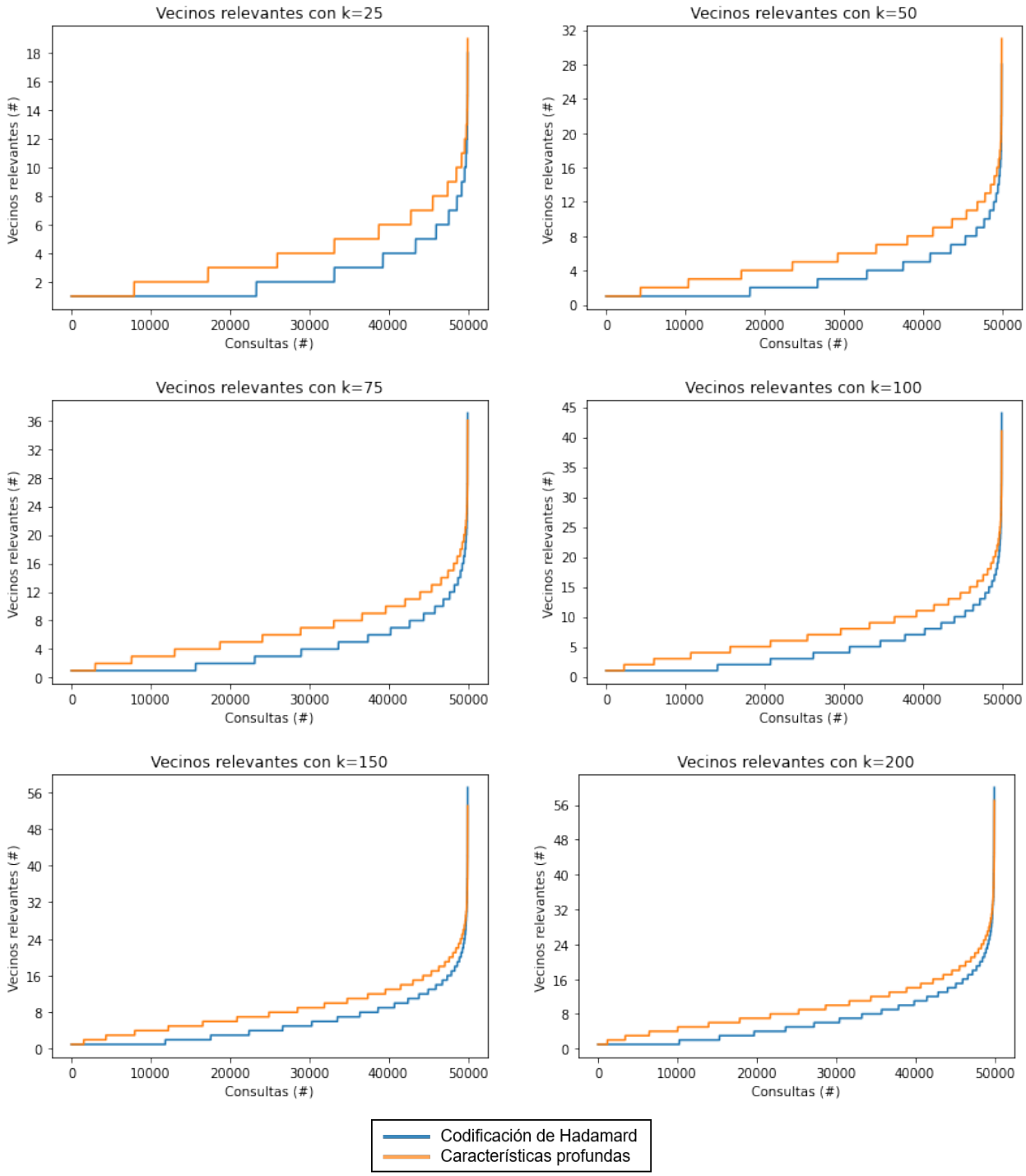


Figura 39. Comparación de vecinos más cercanos de HSP con variaciones de k en ImageNet. La línea naranja representa las características profundas y la línea azul representa la codificación de Hadamard.

Capítulo 6. Discusión

La codificación de Hadamard demostró tener buen desempeño en tareas de navegación y recuperación en las colecciones de ImageNet y Places365. Los resultados en Places365 son mejores que los obtenidos en ImageNet, ya que en ambos clasificadores con un valor de k pequeño, se obtiene que la codificación de Hadamard es mejor que las características profundas en más de la mitad de las clases. Además, al aumentar el valor de k, la variación es muy poca. Por otra parte, el comportamiento en el conjunto de datos de ImageNet es totalmente diferente, ya que el aumentar el valor de k tiene un efecto positivo en la codificación de Hadamard. Al inicio con una k=25 las características profundas son mejores en más de la mitad de las clases. Sin embargo, el aumento en el valor de k provoca que la codificación de Hadamard sea mejor. Esto se pudo ver al seleccionar un valor de k=200, en donde la codificación de Hadamard era mejor en más de 800 clases, y la diferencia en las clases restantes era mínima. Por lo tanto, para el dominio donde se tienen pocas instancias por clase, las características profundas son mejores, ya que obtuvieron buenos resultados en el conjunto de datos ImageNet. Además, con valores de k pequeños, por ejemplo, 25 o 50, siempre fueron mejores que la codificación de Hadamard. Sin embargo, la codificación de Hadamard puede ser mejor si se toma un valor de vecinos relevantes mayor a 100.

El algoritmo HSP demostró un buen rendimiento tanto en las características profundas como en la codificación de Hadamard. Con diferencia a kNN, HSP no requiere indicar un valor de k que determina la cantidad de vecinos más cercanos a utilizar. En cambio, HSP construye una vecindad con elementos semejantes y diversos a una consulta dada. Sin embargo, las gráficas de vecinos relevantes de HSP muestran que una alta cantidad de consultas de ambos conjuntos de datos (Places365 e ImageNet) solamente devuelven un vecino relevante. Aunado a esto, se puede decir que en la mayoría de casos, ese vecino es de «calidad», es decir, que será relevante para el usuario, ya que es similar a la consulta dada o tiene la misma clase que la consulta. Esto último lo podemos ver en la Tabla 3 en donde se muestra la precisión de las consultas que regresan un vecino relevante en ambos conjuntos de datos utilizando las técnicas de la codificación de Hadamard y las características profundas. Se puede ver que excepto cuando se utiliza un valor de $k = 25$ en Places365 con la representación de las características profundas, todos los demás valores de precisión están por

Tabla 3. Precisión del número de consultas en los conjuntos de datos Places365 e ImageNet con la codificación de Hadamard y las características profundas que proporcionan un vecino relevante al usar el algoritmo HSP.

	Places365		ImageNet	
k	precisión de CH	precisión de CF	precisión de CH	precisión de CF
25	58.96 %	48.59 %	66.30 %	63.20 %
50	64.80 %	54.98 %	72.51 %	66.05 %
75	67.27 %	54.80 %	75.55 %	67.81 %
100	68.40 %	54.02 %	77.52 %	69.71 %
150	70.39 %	57.14 %	79.88 %	69.83 %
200	71.60 %	60.82 %	81.52 %	72.57 %

CH: Codificación de Hadamard; CF: Características profundas

encima del 50 %. La precisión aumenta con respecto al crecimiento del valor de *k*. Además, en ambos conjuntos de datos, la precisión es mayor al utilizar la codificación de Hadamard.

Existen dos razones por las cuales ocurre que la consulta solamente tenga un vecino relevante al utilizar el algoritmo HSP. Esto puede ocurrir cuando la consulta es un punto aislado, es decir, que está muy lejano a las imágenes de la colección. Si esto pasa, no importa el valor de *k* que se seleccione, siempre se obtendrá un vecino. Este vecino es el más cercano a la consulta y no será seleccionado otro vecino, ya que al encontrar el vecino más cercano, se realiza un hiperplano que separa en dos el plano y todos los demás puntos quedan en la zona prohibida del grafo HSP. Esto indicaría que la consulta es un valor atípico (del inglés *outlier*), es decir, la consulta no sigue la misma distribución que el resto de los datos; ocasionando que no tenga imágenes semejantes en la colección. Lo anterior lo podemos ver en la Figura 40, en el inciso (a) se muestra el comportamiento de una consulta que se considera un *outlier*, es decir, aunque el valor de *k* aumente, no cambia la cantidad de vecinos relevantes; siempre se mantiene regresando un vecino relevante. En el inciso (b) se muestra de manera gráfica la recuperación de un vecino relevante de una consulta que se considera un *outlier*.

El otro caso sería cuando es necesario aumentar el radio con el que se construye el grafo HSP para recuperar más vecinos relevantes. La Figura 41 muestra el comportamiento de una consulta que se considera un *inlier*. Un *inlier* es un dato que se encuentra en error, es decir, sujeto a errores de medición. Sin embargo, se encuentra

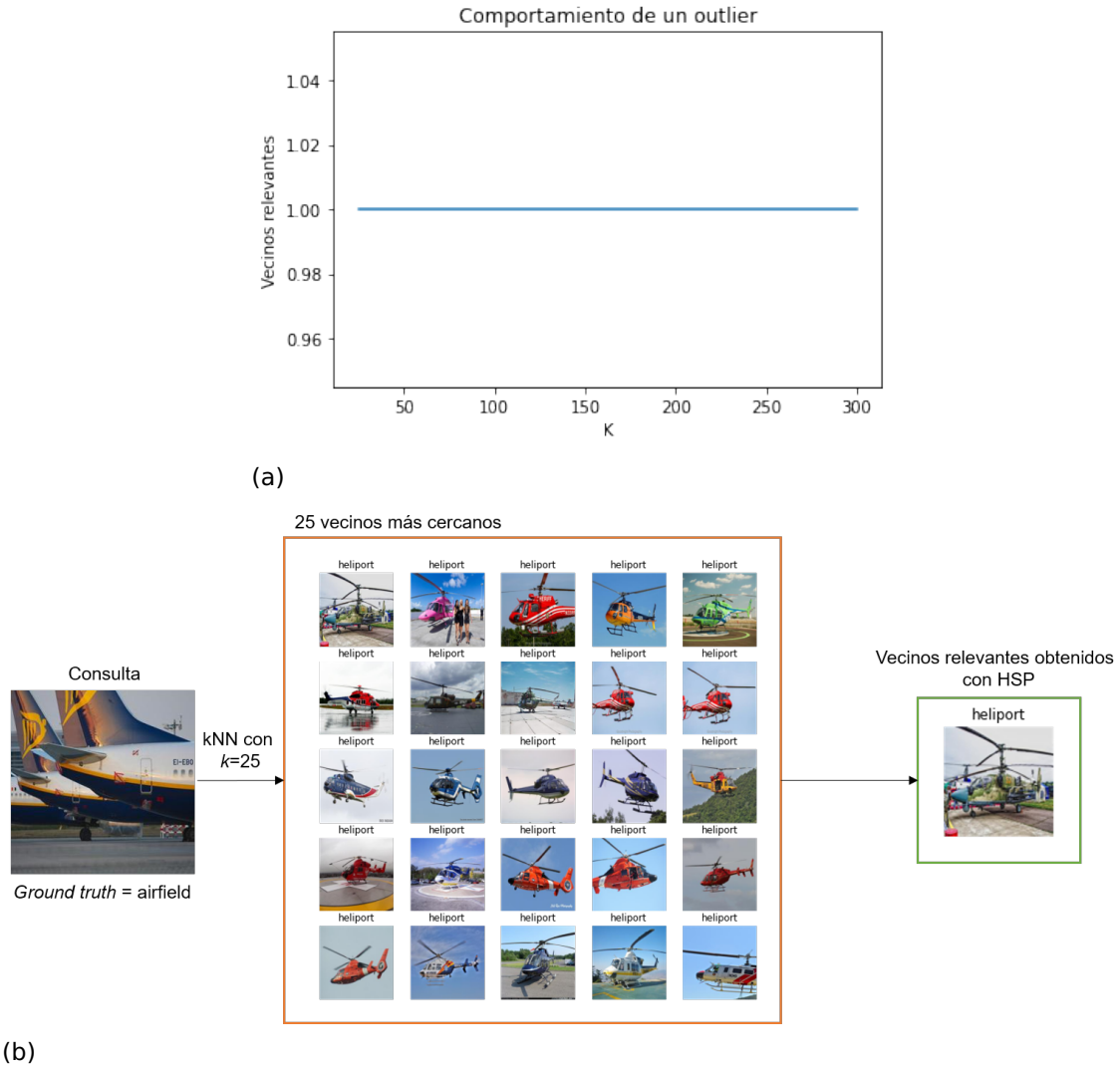


Figura 40. Comportamiento de una consulta que se considera un *outlier* al clasificarla usando el algoritmo HSP. El inciso (a) muestra como al variar el valor de k, siempre se obtiene un vecino relevante, lo cual representa que la consulta es un *outlier*. Mientras que el inciso (b) muestra de manera gráfica la recuperación de un vecino relevante de una consulta que se considera un *outlier*.

dentro de la distribución de los datos. En el inciso (a) de la Figura 41, se puede ver el comportamiento de una consulta que se considera *inlier*, ya que al aumentar el valor de k se consigue recuperar más vecinos relevantes, inicialmente se recuperaba un vecino relevante y al aumentar el valor de k se logran recuperar tres vecinos relevantes. Además, se muestra en el inciso (b) de la Figura 41 un ejemplo gráfico de la recuperación de vecinos relevantes con valores de $k = 25$ y $k = 50$ de una consulta que se considera un *inlier*.

El algoritmo kNN tiene la diferencia con HSP que se debe seleccionar el hiperpa-

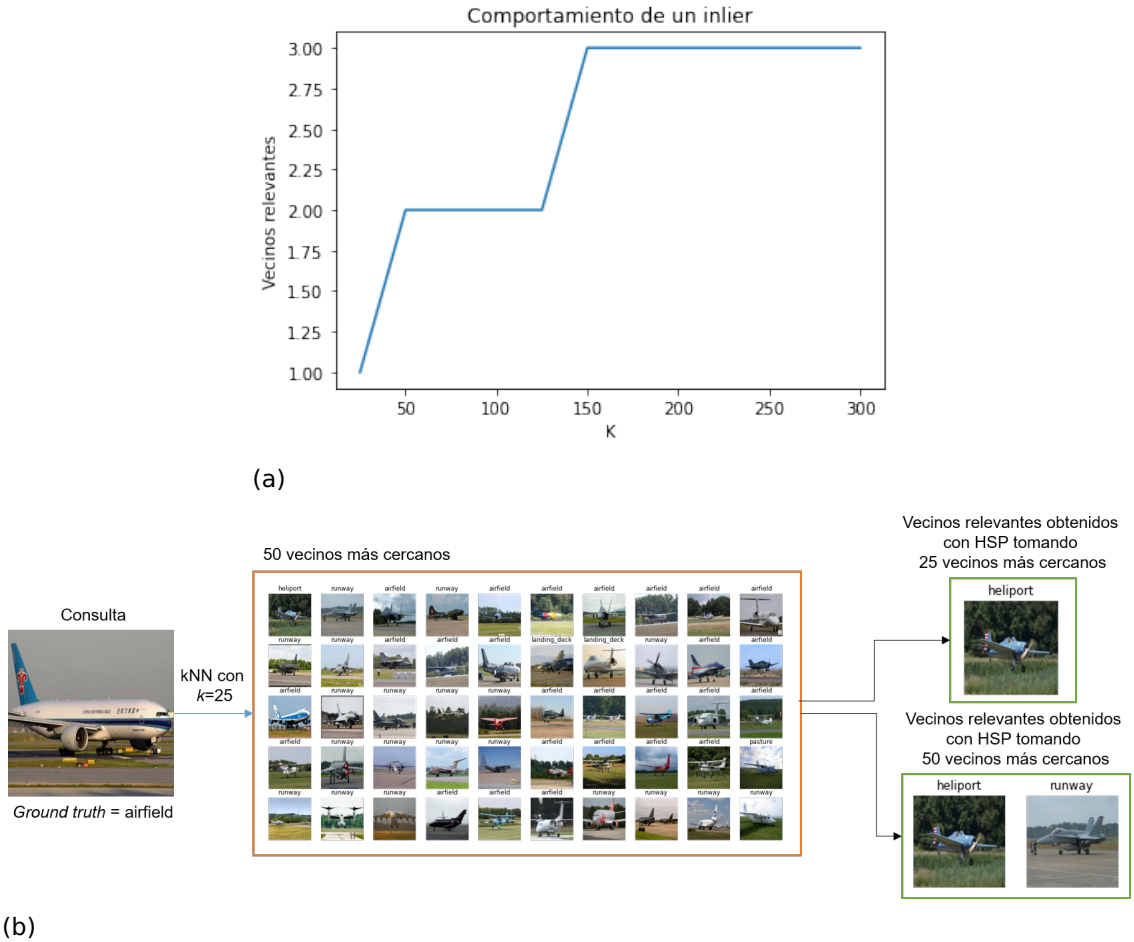


Figura 41. Comportamiento de una consulta que se considera un *inlier* al clasificarla usando el algoritmo HSP. El inciso (a) muestra como al variar el valor de k, se logra aumentar el número de vecinos relevantes recuperados, lo cual representa un *inlier*. Mientras que en el inciso (b) se muestra de manera gráfica la recuperación de vecinos relevantes con valores de $k = 25$ y $k = 50$ de una consulta que se considera un *inlier*.

rámetro k, lo cual resulta ser una tarea compleja y no es óptima para sistemas de recuperación en producción. Mientras que con el algoritmo HSP la cantidad de vecinos cambia dependiendo la consulta. Dado sus propiedades de semejanza y diversidad, este algoritmo resulta ser una buena opción para mostrar al usuario imágenes relevantes, semejantes, y diversas.

Los experimentos realizados con las características profundas y la codificación de Hadamard permitieron conocer la eficacia de los vectores binarios en codificación de Hadamard en tareas de recuperación y navegación. Utilizar la codificación de Hadamard como representación de imágenes de una colección resulta ser una buena opción, ya que debido a la alta dimensionalidad de los vectores de características pro-

Tabla 4. Características del vector de características profundas y el vector binario en codificación de Hadamard. Se muestra el espacio en bits y la dimensión del vector.

	Características profundas		Codificación de Hadamard	
Dataset	dimensión vector	tamaño (bytes)	dimensión vector	tamaño (bytes)
Places365	2048	65,536	512	128
ImageNet	2048	65,536	1024	64

fundas que proporcionan los modelos de redes neuronales, como ResNet o VGG, e indexar miles o millones de imágenes, requiere un alto consumo de memoria principal (RAM). En la Tabla 4 se muestra el tamaño de memoria requerido de los vectores de características profundas y los vectores en codificación de Hadamard en los conjuntos de datos de ImageNet y Places365.

La codificación de Hadamard reduce los valores del vector a la mínima unidad de representación que es el bit, logrando reducir la huella de memoria sin perder relevancia en la clasificación. Dado que la distancia utilizada en la codificación de Hadamard es la distancia de Hamming, es computacionalmente menos costosa que la distancia euclidiana, produciendo que se necesiten menos ciclos de cómputo al recuperar imágenes semejantes. Un pequeño experimento que se realizó para comprobar esto, fue calcular un millón de distancias euclidianas y un millón de distancias de Hamming. El millón de distancias euclidianas fue calculado en 10.357863 segundos. Mientras que el millón de distancias de Hamming fue calculado en tan solo 0.044035 segundos. Lo que representa que se determine que el cálculo de la distancia de Hamming es 200 veces más rápido que el cálculo de la distancia euclidiana.

Los sistemas de recuperación multimedia del estado del arte utilizan la representación con características profundas. Los resultados obtenidos indican mediante las comparaciones que la representación con códigos de Hadamard es en promedio mejor que la representación con características profundas en tareas de recuperación y navegación.

Capítulo 7. Conclusiones y trabajo futuro

Existe una amplia variedad de sistemas de recuperación multimedia en el mercado, como Google Images o Pinterest, sin embargo, la estructura interna de estos sistemas no es pública. La investigación en estos sistemas ha aumentado gracias a competencias como *Video Browser Showdown* (VBS). La constante entre los sistemas del estado del arte presentados en VBS es el uso de modelos de aprendizaje profundo para etiquetar imágenes o videos.

En los últimos años, las redes neuronales han tenido un auge en la investigación de nuevos modelos. Estos modelos han aportado grandes resultados en tareas como la clasificación de imágenes. Los modelos utilizados para realizar clasificación son de gran ayuda para etiquetar colecciones multimedia, ya que permiten extraer de sus capas finales (por lo general, la penúltima capa), un vector representativo de la imagen procesada. Sin embargo, la alta dimensión del vector obtenido se convierte en un obstáculo al trabajar con colecciones multimedia de gran tamaño.

7.1. Conclusiones

En este trabajo se hizo una propuesta para mejorar la representación de imágenes con características profundas mediante el uso de códigos de Hadamard. La implementación de los códigos de Hadamard en arquitecturas de redes neuronales convolucionales es sencilla, únicamente hay que cambiar el hiperparámetro que corresponde al número de clases, eliminar la última capa del modelo, agregar una capa con la función de activación HardTanh, y convertir el vector resultante al procesar una imagen en el modelo a binario.

El vector resultante contiene bits, lo cual trae consigo ventajas como un menor consumo de memoria RAM y requerir menos ciclos de cómputo al calcular la distancia entre vectores, ya que los vectores en codificación de Hadamard utilizan la distancia de Hamming para obtener vectores semejantes. Calcular la distancia entre vectores con la distancia de Hamming es más rápido que calcular la distancia euclidiana, que se utiliza en para calcular distancia entre vectores con características profundas.

También se propuso el uso del grafo HSP para recuperar imágenes relevantes a una

consulta. El objetivo de usar HSP en vez de *kNN* es que no es necesario seleccionar el hiperparámetro k. Además, las propiedades del grafo HSP de construir una vecindad que contenga vecinos semejantes y diversos puede funcionar bien en un sistema de recuperación de imágenes, ya que actúa como un filtrado y se presentan imágenes representativas al usuario.

Un comportamiento muy frecuente al utilizar el algoritmo HSP es que solamente se encontraba un vecino relevante. Esto ocurre porque los k elementos utilizados para construir la vecindad son muy semejantes entre ellos. Por lo cual, el algoritmo HSP únicamente selecciona un representante, es decir, aquel elemento que tenga la menor distancia con la consulta. Los $k-1$ elementos restantes, quedan dentro de la región prohibida, ya que todos los elementos son muy semejantes. Por lo tanto, no existen más elementos para construir el grafo y como resultado se obtiene un vecino relevante. De está manera, el algoritmo HSP elimina redundancia en la recuperación, dando como resultado un vecino semejante y diverso que representa a toda la vecindad.

Los experimentos realizados permitieron evaluar la eficacia de la representación con códigos de Hadamard en tareas de navegación en colecciones multimedia de imágenes. Además, se mantuvo la propiedad que tienen las características profundas de obtener objetos semejantes con el cálculo de la distancia euclidiana. Sin embargo, con la representación de Hadamard, para calcular la distancia entre vectores es la distancia de Hamming.

Se desarrolló un prototipo de un sistema de recuperación de imágenes. En este prototipo se incluyeron diferentes formas de recuperar imágenes, por ejemplo, búsqueda por texto, búsqueda dado un ejemplo, y retroalimentación positiva. Para construir el prototipo se realizaron tres etapas: (1) extracción de características del conjunto de imágenes, (2) desarrollo del *backend*, y (3) desarrollo del frontend. Después de construir el prototipo, se realizó una pequeña investigación para conocer las diferentes herramientas que provee Google Cloud y fueron seleccionadas aquellas que pudieran utilizarse en la infraestructura del sistema. Para finalizar, se realizó un despliegue a Google Cloud para probar diferentes herramientas como Google Cloud Storage y Compute Engine. El desarrollo de este prototipo de un sistema de recuperación de imágenes, permitirá ser un punto de partida a futuras investigaciones.

La recuperación de imágenes es una tarea compleja, ya que no es fácil comprender la necesidad del usuario y es por ello que es necesario investigar métodos de indexamiento, recuperación, y herramientas para realizar consultas. Sin lugar a dudas, desarrollar un sistema de recuperación requiere conocimiento en múltiples áreas, como recuperación de información, visión por computadora, desarrollo web, entre otras áreas.

En general, los códigos de Hadamard y el grafo de semi-espacios proximales permiten recuperar imágenes de una manera eficiente, abriendo la posibilidad de realizar búsqueda dado un ejemplo. La combinación de códigos de Hadamard y el grafo HSP fue implementada con éxito en el prototipo del sistema de recuperación de imágenes. Además, con los experimentos realizados pudimos constatar que la representación de imágenes con codificación de Hadamard es en promedio mejor que la representación de imágenes con características profundas.

7.2. Trabajo futuro

El trabajo futuro del proyecto se encuentra enfocado principalmente en el prototipo del sistema de recuperación de imágenes desarrollado. Por ahora, se tiene un prototipo funcional al cual pueden realizarse diferentes pruebas, como pruebas de usabilidad, experiencia de usuario, diseño, entre otras pruebas que estén enfocadas en la interacción del usuario.

El modelo para realizar consultas por texto fue construido simplemente para demostrar cómo funcionaría la búsqueda por texto en la retroalimentación positiva. Sin embargo, este apartado tiene un gran potencial que abre un sinfín de oportunidades de investigación. Será necesario profundizar en el área de Procesamiento del Lenguaje Natural para estudiar a detalle la interacción entre la computadora y el lenguaje humano. Dentro de está área existen redes neuronales que sería buena idea estudiarlas, de manera que se diseñe y desarrolle un modulo de búsqueda por texto más eficiente y robusto.

Uno de los modelos que podría estudiarse es la representación de codificador bidireccional de transformadores (BERT, del inglés *Bidirectional Encoder Representations from Transformers*) (Devlin *et al.*, 2019). Este modelo puede interpretar el contexto

completo de una palabra analizando las palabras que vienen antes y después, lo que resulta muy útil para comprender la intención de búsqueda que tiene el usuario al realizar una consulta por texto.

BERT intenta comprender el lenguaje humano, descubrir lo que está buscando el usuario, y presentar información que sea relevante para el usuario. En pocas palabras, BERT permite entender a profundidad el contexto y la temática de toda la frase que introduce el usuario como consulta.

Con lo anterior, se tendría un sistema de recuperación de imágenes completo y sería interesante poder participar en la competencia *Video Browser Showdown* para conocer cómo se desempeña el sistema de recuperación en un ambiente que sirve como marco de referencia. Lograr participar abriría la puerta a conocer otros enfoques que utilizan investigadores del área y recibir retroalimentación por parte de expertos en la materia de sistemas de recuperación multimedia.

Literatura citada

Alemu, Y., Koh, J.-b., Ikram, M., y Kim, D.-K. (2009). Image Retrieval in Multimedia Databases: A Survey. En: *2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. pp. 681–689.

Andoni, A., Indyk, P., y Razenshteyn, I. P. (2018). Approximate Nearest Neighbor Search in High Dimensions. *arXiv preprint arXiv: 1806.09823*. Recuperado el 14 de junio de 2021 de: `https://arxiv.org/pdf/1806.09823.pdf`.

Azizpour, H., Razavian, A. S., Sullivan, J., Maki, A., y Carlsson, S. (2015). From generic to specific deep representations for visual recognition. En: *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. pp. 36–45.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., y Yuille, A. L. (2018). DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **40**(4): 834–848.

Chávez, E., Dobrev, S., Kranakis, E., Opatrny, J., Stacho, L., Tejeda, H., y Urrutia, J. (2005). Half-Space Proximal: A New Local Test for Extracting a Bounded Dilation Spanner of a Unit Disk Graph. 12. Vol. 3974, pp. 235–245.

Datta, R., Joshi, D., Li, J., y Wang, J. Z. (2008). Image Retrieval: Ideas, Influences, and Trends of the New Age. *ACM Comput. Surv.*, **40**(2).

Dauzon, S., Bendoraitis, A., y Ravindran, A. (2017). *Django: Web Development with Python: Web Development with Python*. Packt Publishing, primera edición.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., y Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. En: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. pp. 248–255.

Deng, L. (2012). The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine*, **29**(6): 141–142.

Devlin, J., Chang, M.-W., Lee, K., y Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. En: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jun, Minneapolis, Minnesota. Association for Computational Linguistics, pp. 4171–4186.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., y Darrell, T. (2014). DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. ICML'14, p. 647–655.

Duane, A., Þór Jónsson, B., y Gurrin, C. (2020). VRLE: Lifelog Interaction Prototype in Virtual Reality: Lifelog Search Challenge at ACM ICMR 2020. En: *Proceedings of the Third Annual Workshop on Lifelog Search Challenge*, New York, NY, USA. Association for Computing Machinery, LSC '20, p. 7–12.

Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., Villena-Martinez, V., y Rodríguez, J. G. (2017). A Review on Deep Learning Techniques Applied to Semantic Segmentation. *arXiv preprint arXiv: 1704.06857*. Recuperado el 13 de mayo de 2021 de: `https://arxiv.org/pdf/1704.06857.pdf`.

Girshick, R. (2015). Fast R-CNN. En: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December.

Girshick, R., Donahue, J., Darrell, T., y Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

Godbolt, M. (2016). *Frontend Architecture for Design Systems: A Modern Blueprint for Scalable and Sustainable Websites*. O'Reilly Media, Inc, primera edición.

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep Learning*. MIT Press. Recuperado el 15 de enero de 2021 de: `http://www.deeplearningbook.org`.

Gurrin, C., Jónsson, B., Schöffmann, K., Dang-Nguyen, D.-T., Lokoč, J., Tran, M.-T., Hürst, W., Rossetto, L., y Healy, G. (2021). Introduction to the Fourth Annual Lifelog Search Challenge, LSC'21. New York, NY, USA. Association for Computing Machinery, ICMR '21, p. 690–691.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., y Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, **585**(7825): 357–362.

He, K., Zhang, X., Ren, S., y Sun, J. (2016). Deep Residual Learning for Image Recognition. En: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 770–778.

He, K., Gkioxari, G., Dollar, P., y Girshick, R. (2017). Mask R-CNN. En: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct.

Hervé, N. y Boujemaa, N. (2007). Image annotation: which approach for realistic databases? 07. pp. 170–177.

Hoyos, A., Ruiz, U., y Chavez, E. (2021). Hadamard's Defense Against Adversarial Examples. *IEEE Access*, **9**: 118324–118333.

Huang, T. S., Dagli, C. K., Rajaram, S., Chang, E. Y., Mandel, M. I., Poliner, G. E., y Ellis, D. P. W. (2008). Active Learning for Interactive Multimedia Retrieval. *Proceedings of the IEEE*, **96**(4): 648–667.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, **9**(3): 90–95.

Indyk, P. y Motwani, R. (1998). Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. En: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, New York, NY, USA. Association for Computing Machinery, STOC '98, p. 604–613.

Jiao, L., Zhang, F., Liu, F., Yang, S., Li, L., Feng, Z., y Qu, R. (2019). A Survey of Deep Learning-Based Object Detection. *IEEE Access*, **7**: 128837–128868.

Jing, Y., Liu, D., Kislyuk, D., Zhai, A., Xu, J., Donahue, J., y Tavel, S. (2015). Visual Search at Pinterest. En: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA. Association for Computing Machinery, KDD '15, p. 1889–1898.

Jégou, H., Douze, M., y Schmid, C. (2011). Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **33**(1): 117–128.

Jónsson, B., Khan, O. S., Koelma, D. C., Rudinac, S., Worring, M., y Zahálka, J. (2020). Exquisitor at the Video Browser Showdown 2020. En: Y. M. Ro, W.-H. Cheng, J. Kim, W.-T. Chu, P. Cui, J.-W. Choi, M.-C. Hu, y W. De Neve (eds.), *MultiMedia Modeling*, Cham. Springer International Publishing, pp. 796–802.

Khan, O. S. (2020). An Interactive Learning System for Large-Scale Multimedia Analytics. En: *Proceedings of the 2020 International Conference on Multimedia Retrieval*, New York, NY, USA. Association for Computing Machinery, ICMR 20, p. 368–372.

Khan, O. S., Jónsson, B., Larsen, M., Poulsen, L., Koelma, D. C., Rudinac, S., Worring, M., y Zahálka, J. (2021). Exquisitor at the Video Browser Showdown 2021: Relationships Between Semantic Classifiers. En: J. Lokoč, T. Skopal, K. Schoeffmann, V. Mezaris, X. Li, S. Vrochidis, y I. Patras (eds.), *MultiMedia Modeling*, Cham. Springer International Publishing, pp. 410–416.

Kirillov, A., He, K., Girshick, R., Rother, C., y Dollar, P. (2019). Panoptic Segmentation. En: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J. B., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., y Team, J. D. (2016). Jupyter Notebooks - a publishing format for reproducible computational workflows. En: *ELPUB*.

Kratochvil, M., Mejzlík, F., Veselý, P., Soućek, T., y Lokoć, J. (2020). SOMHunter: Lightweight Video Search System with SOM-Guided Relevance Feedback. En: *Proceedings of the 28th ACM International Conference on Multimedia*, New York, NY, USA. Association for Computing Machinery, MM '20, p. 4481–4484.

Kratochvíl, M., Veselý, P., Mejzlík, F., y Lokoč, J. (2020). SOM-Hunter: Video Browsing with Relevance-to-SOM Feedback Loop. En: Y. M. Ro, W.-H. Cheng, J. Kim, W.-T. Chu, P. Cui, J.-W. Choi, M.-C. Hu, y W. De Neve (eds.), *MultiMedia Modeling*, Cham. Springer International Publishing, pp. 790–795.

Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. En: F. Pereira, C. J. C. Burges, L. Bottou, y K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc., Vol. 25.

Lampert, C., Blaschko, M., y Hofmann, T. (2008). Beyond sliding windows: Object localization by efficient subwindow search. 07. pp. 1 – 8.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., y Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, **1**(4): 541–551.

Lew, M. S., Sebe, N., Djeraba, C., y Jain, R. (2006). Content-Based Multimedia Information Retrieval: State of the Art and Challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, **2**(1): 1–19.

Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., y Lin, X. (2020). Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering*, **32**(8): 1475–1488.

Lokoč, J., Kovalčík, G., y Souček, T. (2020). VIRET at Video Browser Showdown 2020. En: Y. M. Ro, W.-H. Cheng, J. Kim, W.-T. Chu, P. Cui, J.-W. Choi, M.-C. Hu, y W. De Neve (eds.), *MultiMedia Modeling*, Cham. Springer International Publishing, pp. 784–789.

Long, J., Shelhamer, E., y Darrell, T. (2015). Fully Convolutional Networks for Semantic Segmentation. En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

Malkov, Y., Ponomarenko, A., Logvinov, A., y Krylov, V. (2014). Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, **45**: 61–68.

Malkov, Y. A. y Yashunin, D. A. (2020). Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **42**(4): 824–836.

Muja, M. y Lowe, D. G. (2014). Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **36**(11): 2227–2240.

Nadareishvili, I., Mitra, R., McLarty, M., y Amundsen, M. (2016). *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, Inc, primera edición.

Nguyen, P. A., Wu, J., Ngo, C.-W., Francis, D., y Huet, B. (2020). VIREO @ Video Browser Showdown 2020. En: Y. M. Ro, W.-H. Cheng, J. Kim, W.-T. Chu, P. Cui, J.-W. Choi, M.-C. Hu, y W. De Neve (eds.), *MultiMedia Modeling*, Cham. Springer International Publishing, pp. 772–777.

Pan, S. J. y Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, **22**(10): 1345–1359.

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

Peška, L., Kovalčík, G., Souček, T., Škrhák, V., y Lokoč, J. (2021). W2VV++ BERT Model at VBS 2021. En: J. Lokoč, T. Skopal, K. Schoeffmann, V. Mezaris, X. Li, S. Vrochidis, y I. Patras (eds.), *MultiMedia Modeling*, Cham. Springer International Publishing, pp. 467–472.

Ragnarsdóttir, H., Þorleiksdóttir, T., Khan, O. S., Jónsson, B. T., Guðmundsson, G. T., Zahálka, J., Rudinac, S., Amsaleg, L., y Worring, M. (2019). Exquisitor: Breaking the Interaction Barrier for Exploration of 100 Million Images. En: *Proceedings of the 27th ACM International Conference on Multimedia*, New York, NY, USA. Association for Computing Machinery, MM '19, p. 1029–1031.

Razavian, A. S., Azizpour, H., Sullivan, J., y Carlsson, S. (2014). CNN Features Off-the-Shelf: An Astounding Baseline for Recognition. En: *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. pp. 512–519.

Redmon, J. y Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. En: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 6517–6525.

Redmon, J. y Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *arXiv preprint arXiv: 1804.02767*. Recuperado el 2 de febrero de 2021 de: http://arxiv.org/abs/1804.02767.

Redmon, J., Divvala, S., Girshick, R., y Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

Reed, I. S. y Solomon, G. (1960). Polynomial Codes Over Certain Finite Fields. *Journal of The Society for Industrial and Applied Mathematics*, **8**: 300–304.

Ren, S., He, K., Girshick, R., y Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **39**(6): 1137–1149.

Schoeffmann, K. (2019). Video Browser Showdown 2012-2019: A Review. En: *2019 International Conference on Content-Based Multimedia Indexing (CBMI)*. pp. 1–4.

Schoeffmann, K., Lokoč, J., y Bailer, W. (2021). 10 Years of Video Browser Showdown. New York, NY, USA. Association for Computing Machinery, MMAsia '20.

Simonyan, K. y Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv: 1409.1556*. Recuperado el 11 de marzo de 2021 de: http://arxiv.org/abs/1409.1556.

Srivastava, P. y Khan, R. (2018). A Review Paper on Cloud Computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, **8**: 17.

Sulyman, S. (2014). Client-Server Model. *IOSR Journal of Computer Engineering*, **16**: 57–71.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., y Rabinovich, A. (2015). Going deeper with convolutions. En: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 1–9.

Talamantes, A. y Chávez, E. (2021). Instance-based learning using the Half-Space Proximal Graph. *arXiv preprint arXiv: 2102.02755*. Recuperado el 21 de febrero de 2021 de: https://arxiv.org/abs/2102.02755.

Tanner, K. (2002). Chapter 7 - Experimental research designs. En: K. Williamson, A. Bow, F. Burstein, P. Darke, R. Harvey, G. Johanson, S. McKemmish, M. Oosthuizen, S. Saule, D. Schauder, G. Shanks, y K. Tanner (eds.), *Research Methods for Students, Academics and Professionals (Second Edition)*. Chandos Publishing, second edition edición, Topics in Australasian Library and Information Studies, pp. 125–146.

Thomee, B., Shamma, D. A., Friedland, G., Elizalde, B., Ni, K., Poland, D., Borth, D., y Li, L.-J. (2016). YFCC100M: The New Data in Multimedia Research. *Commun. ACM*, **59**(2): 64–73.

Valueva, M., Nagornov, N., Lyakhov, P., Valuev, G., y Chervyakov, N. (2020). Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, **177**: 232–243.

Wes McKinney (2010). Data Structures for Statistical Computing in Python. En: Stéfan van der Walt y Jarrod Millman (eds.), *Proceedings of the 9th Python in Science Conference*. pp. 56 – 61.

Wicker, S. B. y Bhargava, V. K. (1999). *Reed-Solomon Codes and Their Applications*. John Wiley & Sons.

Zahálka, J., Rudinac, S., Jónsson, B., Koelma, D. C., y Worring, M. (2018). Blackthorn: Large-Scale Interactive Multimodal Learning. *IEEE Transactions on Multimedia*, **20**(3): 687–698.

Zhai, A., Kislyuk, D., Jing, Y., Feng, M., Tzeng, E., Donahue, J., Du, Y. L., y Darrell, T. (2017). Visual Discovery at Pinterest. En: *Proceedings of the 26th International Conference on World Wide Web Companion*, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee, WWW '17 Companion, p. 515–524.

Zhang, D., Islam, M. M., y Lu, G. (2012). A review on automatic image annotation techniques. *Pattern Recognition*, **45**(1): 346–362.

Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., y Torralba, A. (2018). Places: A 10 Million Image Database for Scene Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **40**(6): 1452–1464.

Zhu, M. y Wu, Y.-F. B. (2014). Search by Multiple Examples. En: *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, New York, NY, USA. Association for Computing Machinery, WSDM '14, p. 667–672.