Universidad Zaragoza
1542

Trabajo Fin de Grado

Sistema de indexado y búsqueda multimodal en archivos multimedia

Multimodal indexing and search system in multimedia files

Autora

María García Cutando

Director

Eduardo Lleida Solano

Titulación del autor

Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2023

Sistema de indexado y búsqueda multimodal en archivos multimedia

RESUMEN

Los sistemas de indexado y búsqueda son herramientas que permiten analizar archivos multimedia utilizando diferentes modalidades de entrada. Estos sistemas suelen utilizar técnicas de procesamiento y aprendizaje automático.

Uno de los principales retos en el desarrollo de sistemas de indexado y búsqueda multimodal, es la diversidad y complejidad de los archivos multimedia. Disponen de diferentes propiedades, como tamaños y formatos, además de otras características relacionadas con el contenido de estos.

En los últimos años, debido al incremento de demanda de archivos multimedia, es necesario disponer de sistemas que gestionen el contenido de la forma más eficiente posible. Encontrando y recuperando los datos más importantes que los identifiquen. Para ello, el auge de las redes neuronales ha permitido que, mediante entrenamiento, se puedan extraer metadatos útiles. Permitiendo efectuar así, búsquedas de objetos, textos, rostros o incluso contextos.

Los sistemas de indexado permiten a la vez, obtener una visión de los datos ordenada, haciendo posible la agrupación o clasificación por contenidos similares. Es el caso de la segmentación de vídeos por cambios de plano. Gracias a estas nuevas tecnologías, se ha mejorado en la velocidad y precisión de procesamiento y recuperación de los datos.

Este trabajo de Fin de Grado aborda este problema, haciendo uso de técnicas de inteligencia artificial, para extraer y procesar la información más relevante. Creando un sistema de indexado y búsqueda multimodal, mediante la utilización del modelo de entrenamiento *zero-shot* CLIP de OpenAI. El sistema organiza la información y la almacena en la base de datos vectorial Milvus, para realizar fácilmente búsquedas posteriores. En consecuencia, se ha evaluado la creación de un sistema de segmentación semántica por escenas. Y así, conseguir comprimir la gran cantidad de información procesada, reduciendo el coste computacional a la hora de la realización de búsquedas. Todo ello se ha conseguido combinando la extracción de fotogramas junto a comparativas vectoriales. El resultado proviene del cálculo de la distancia coseno entre vectores y el análisis del gradiente resultante.

A lo largo de este documento se describen las tecnologías y pruebas realizadas para la creación del sistema. Un sistema de indexado de video, búsqueda multimodal, segmentación y compresión, que ayuda a trabajar con una gran masa de archivos multimedia de una manera orientativa y simple.

Índice de contenidos

Índice de figuras

Glosario de términos

CLIP. *Contrastive Language–Image Pre-training*
OpenCV. *Open-Source Computer Vision Library*
NLP. *Natural Language Processing*
RGB. *Red, Green, and Blue*
E-M. *Expectation-Maximization*
ISR. *Information Search and Retrieval*
SQL. *Structured Query Language*
NoSQL. *Not only Structured Query Language*
ML. *Machine Learning*
WSL. *Windows Subsystem for Linux*
ANN. *Approximate Nearest Neighbor*

Capítulo 1

Introducción

Este primer bloque se dedicará a dar una breve explicación de la motivación de este proyecto. Además de los objetivos y funcionalidades que quieren alcanzarse en el desarrollo de este. Para finalmente resolver las posibles problemáticas que se desean plantear.

1.1. Motivación

La producción digital ha cambiado el ciclo de producción de muchos organismos como la televisión, permitiendo a los usuarios acceder al contenido almacenado por el servicio de registros en dispositivos digitales. De esta forma, además de editar, transmitir y procesar la información, pueden buscarla, organizarla y acceder a ella de forma rápida y sencilla.

Los nuevos formatos y la publicación del contenido por diversas instituciones, junto al creciente uso de Internet, han hecho que no solo sea importante que los propietarios accedan al contenido. Sino que también se ha creado una necesidad creciente para los usuarios finales, de acceder a la información audiovisual en un formato fácil de entender. Un formato que le permita explorar fragmentos de contenido de interés de forma no secuencial.

A medida que se ha ampliado la gama de aplicaciones, la cantidad y la variedad del contenido multimedia, ha aumentado el interés de investigación en esta área.

> No es un secreto que gran cantidad de la información que manejan actualmente las empresas se encuentra organizada en archivos y documentos no estructurados. Debido a ello, un gran número de organizaciones han tomado conciencia de que la consolidación, acceso y procesamiento de datos no estructurados es un factor importante para la optimización y el análisis de los procesos empresariales. (Fernández, Miranda, Guerrero, & Piccoli, 2010) [16].

Como se puede observar en la Figura 1 [11], se ha cumplido lo que predijo Cisco, se ha generado un increíble crecimiento en la creación, distribución y

consumo de contenido audiovisual durante la última década. Esto ha impulsado el desarrollo de técnicas automatizadas de análisis de video para describir y organizar la información. El desarrollo de estos sistemas facilita el desarrollo de algoritmos de búsqueda más rápidos y precisos.

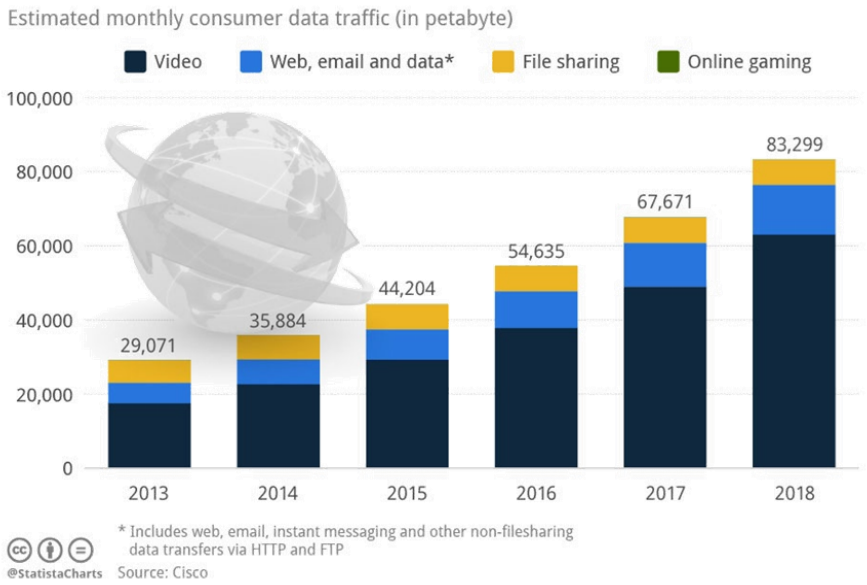

Figura 1: Evolución del consumo de material multimedia.

El análisis del contenido de estos archivos es una tarea fundamental para promover dicho acceso. Esta tarea compleja ha sido tradicionalmente realizada manualmente. Profesionales de la temática se encargaban de asignar metadatos a las unidades documentales. Es decir, a las distintas emisiones de video, se les extraían metadatos que se consideraban potencialmente útiles para su uso posterior.

Los cambios generalizados en el soporte digital y la mayor eficiencia del procesamiento automatizado han permitido desarrollar herramientas innovadoras para segmentar vídeos, e indexarlos. Es decir, seleccionar fotogramas representativos y ordenarlos según ciertas características, para finalmente presentarlos a los usuarios y facilitar la exploración y la recuperación eficiente de los datos visuales, ahorrando así gran esfuerzo y tiempo en etiquetar automáticamente los fotogramas o escenas.

La tecnología ha evolucionado rápida y notablemente, aunque por contra, se ha aumentado y seguirá creciendo exponencialmente la variedad y cantidad de información a procesar. Por lo que existe un compromiso destacable entre novedad y dificultad.

El desarrollo de contenido multimedia ha incluido una dificultad extra en el almacenamiento de este tipo de datos. Se trata de archivos de gran tamaño cuya

información es intrínseca en cada recurso, por lo que resulta una tarea compleja. Por tanto, es necesario disponer de sistemas que gestionen el contenido audiovisual, y buscar un nuevo enfoque que permita la indexación de documentos multimedia de manera simple y automática.

Planteada la problemática es preciso elegir las herramientas adecuadas para el desarrollo del proyecto. Se ha realizado una búsqueda bibliográfica acerca del tema propuesto, y a partir de esa base se han evaluado las posibilidades de los diferentes mecanismos y algoritmos que podrían llegar a ser válidos. Se han elegido unas tecnologías que ofrecían buenos resultados teniendo en cuenta los requisitos del trabajo. Estos requisitos planteaban la idea de usar un modelo multimodal que admitiera inserción de elementos de entrada de diferentes modalidades. También se requería que en el sistema fuera posible la comparación entre ellos dentro de un mismo dominio. Esto permitiría hacer búsquedas de objetos, rostros o textos, o incluso reconocer acciones dentro de vídeos.

El sistema tiene que recoger la información indexada para poder localizar los datos de una manera ordenada. Y como requisito final, debe permitir relacionar los propios fotogramas de manera global, para segmentar los vídeos por planos desde un enfoque natural. Esto último hace referencia a que el resultado se asemeje al realizado por una persona manualmente.

En el artículo *Modern Information Retrieval: A Brief Overview* (Singhal, 2001) [42], se muestran los avances en el campo de la búsqueda de información. Con este proyecto se ha querido seguir con el estudio de la búsqueda y recuperación de información, centrándose en el uso de archivos de multimedia, particularmente en el análisis de video.

Seguirá una serie de procesos empezando por la identificación del tipo de dato a evaluar y el modelaje de estos. Seguidos por la elección de almacenaje del contenido teniendo en cuenta los propósitos planteados. Una vez, elegido el mecanismo para tratar los datos, se precederá con la extracción de metadatos. De ahí, se evaluarán los métodos de segmentación existentes para conseguir aprovechar los recursos al máximo. Y finalmente, se indexarán las escenas para la realización de búsquedas.

En definitiva, el motivo de este proyecto viene de la necesidad de llegar a resolver la gestión de la multimedia reduciendo costes y acelerando los procesos. Buscando un procedimiento óptimo que ayude no solo a alcanzar unos objetivos, sino a estudiar este tipo de datos.

1.2. Objetivos

El objetivo principal de este trabajo consiste en evaluar las posibilidades de indexación y búsqueda de archivos multimedia. Se utilizará la base de datos de vectores Milvus en conjunto con el modelo *zero-shot*, basado en una red neuronal de entrenamiento contrastivo CLIP de OpenAI. En la Figura 2, se encuentran los logos de ambas tecnologías [45] [12].

Figura 2: Herramientas principales utilizadas en el desarrollo del proyecto.

> Durante las manipulaciones humanas y robóticas, nos enfrentamos al desafío de tener que interpretar una gran cantidad de datos visuales en un corto periodo de tiempo. Los datos de los sensores deben estructurarse de manera que la información visual relevante para la tarea sea más accesible. El reconocimiento de los objetos y el contexto de la escena de una manera consistente en el tiempo juega un papel central (Husain, 2016) [19].

Por ende, el reto es crear un sistema que permita muestrear a una frecuencia concreta dada los fotogramas de los distintos vídeos introducidos en el sistema. Extrayendo y procesando la información más relevante de cada uno de ellos. Se desea definir cuáles son los elementos importantes que se deben describir, cómo agrupar los fotogramas para una descripción semántica, y a partir de esta definición, fragmentar los vídeos. Esta selección de escenas permitirá juntar grupos de imágenes relacionadas entre sí, siendo estas indexadas y organizadas.

Finalmente, se quiere almacenar los segmentos en la base de datos Milvus. Incluyendo en el esquema de las colecciones los siguientes atributos: un identificador y diferentes metadatos. Entre los cuales se encuentra el vector denso (*embedding*) que representa la toma y contiene las características de esta, su tiempo de aparición, es decir, el intervalo de fotogramas que componen la escena estimada, y su ubicación, esto es, la localización del video dentro del lugar de ejecución del programa. Por último, el sistema debe permitir búsquedas multimodales gracias a la representación vectorial almacenada en Milvus, permitiendo comparar el texto de entrada con las imágenes de la base de datos.

Como objetivo en el futuro, se plantea que el sistema tenga como aplicación la búsqueda de cada segmento, para crear nuevas composiciones en base a antiguas tomas.

En cuanto a las herramientas que se van a utilizar, el sistema está implementado principalmente gracias a la base de datos Milvus y la red neuronal CLIP.

Se ha utilizado el lenguaje de programación Python para el desarrollo del proyecto. “El lenguaje debe ser extensible de modo que algoritmos con dominio específico puedan coexistir con algoritmos con funcionalidad genérica.” (Fernández, Miranda, Guerrero, & Piccoli, 2010) [16]. La librería OpenCV se usará para el tratamiento de los datos. Y en relación con los procesos que componen el desarrollo del trabajo, se ha seguido la estructura de la Figura 3. Compuesta por 5 pasos, los dos primeros sirven para el modelado de los datos y los tres últimos corresponden a las tareas principales: segmentación, indexación y búsqueda.

Figura 3: Fases del proyecto.

El proceso de indexado incluirá una indexación por *frames*, la información correspondiente a los fotogramas en su totalidad. Cada video se guardará en colecciones Milvus distintas, para realizar búsquedas separadas y estudiar su comportamiento en diferentes contextos.

Para la tarea de búsqueda, se convertirá texto en *embedding vectors*, para poder contrastar con los vectores que representan los fotogramas extraídos en el indexado e importados en la base de datos.

Por último, se aprovechará la extracción de fotogramas a frecuencia mínima de muestreo. La frecuencia de muestreo será la del propio video, para conseguir la mayor precisión posible.

Las capturas de fotogramas servirán como entrada para la implementación de la tarea de segmentación. Permitirá identificar el comienzo y el final de una escena de video utilizando un método de comparación entre vectores, basado en el cálculo de la similitud entre pares. Para distinguir la existencia de un cambio de plano, se hará uso de la distancia coseno y del gradiente en una de las direcciones.

El sistema de segmentación reducirá el tamaño de la base de datos. Mantendrá un único *embedding vector* para un grupo de fotogramas relacionados

entre sí. Junto al rango de elementos de imagen que están identificados por el vector en cuestión.

En resumen, con este proyecto se pretende crear un sistema que permita analizar archivos audiovisuales y localice las relaciones entre sus datos. Con la finalidad de realizar tareas de búsqueda al menor coste posible. De modo que, para crear esta aplicación, se han usado las diferentes herramientas disponibles, eligiendo razonadamente el uso de cada una.

1.3. Estructura de la memoria

La memoria se compone de los siguientes capítulos:

- **Capítulo 1**: Introducción. Se presenta una breve explicación del estado actual del tema propuesto, la novedad que aportan las tecnologías empleadas, y los objetivos del proyecto.

- **Capítulo 2**: Modelo multimodal CLIP. Proporciona una explicación sobre la red neuronal utilizada para el tratamiento de los archivos multimedia, y sus beneficios respecto a otras tecnologías. Incluye el estudio de los tipos de segmentación existentes.

- **Capítulo 3**: Base de datos Milvus. Reúne las especificaciones de la base de datos de vectores de búsqueda, y las ventajas proporcionadas por este tipo de registros en contraste al uso de otros métodos de almacenamiento.

- **Capítulo 4**: Pruebas y Resultados. Recoge los experimentos realizados durante el desarrollo del proyecto con el objetivo de evaluar la funcionalidad del sistema global.

- **Capítulo 5**: Conclusiones y Líneas futuras. Expone los resultados finales obtenidos del proyecto y proporciona algunos enfoques futuros de aplicación del sistema.

Capítulo 2

Modelo multimodal CLIP

En este proyecto, se estudiarán las posibilidades que ofrece CLIP en tareas de indexado, búsqueda y segmentación de video frente a otros modelos. Y la razón de porque se ha escogido CLIP.

2.1. Modelado de datos

Existen muchas alternativas para el modelado de datos dentro de la indexación, búsqueda y segmentación en vídeos. Las opciones más conocidas son:

2.1.1. Modelos basados en reglas

Estos modelos utilizan conjuntos de reglas lógicas o condicionales predefinidas para realizar tareas específicas. Estas reglas se utilizan para describir cómo debe comportarse el sistema ante ciertas condiciones o situaciones. En la Figura 4 [10], se dispone de un ejemplo de modelado de la información basado en un conjunto de reglas.

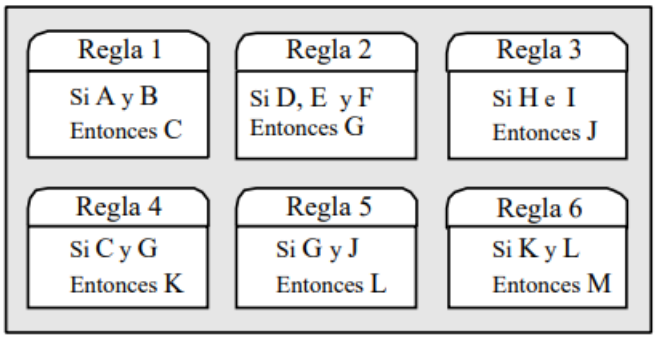

Figura 4: Modelo basado en reglas.

Son efectivos para implementar tareas simples y bien definidas, donde es importante ser explícito acerca de cómo se está tomando una decisión. Son fáciles de entender, razonar e implementar, incluso pueden llegar a ser más rápidos en comparación con otros modelos.

Sin embargo, requieren un gran esfuerzo para definir las reglas y resultan menos flexibles porque dependen completamente de las reglas predefinidas. No

pueden aprender por sí mismos de los datos. En algunas situaciones, si las especificaciones son demasiado concretas, pueden no ser aplicables.

Su rendimiento puede ser limitado, ya que si se quieren ejecutar tareas con finalidades distintas como es el caso de este proyecto, habría que reajustar las reglas para cada función en específico. Además, no sería posible generalizar el sistema para procesar un conjunto de vídeos totalmente diferentes entre sí. Ya que cada archivo, contendrá contenidos totalmente variados que se adecuarán a reglas concretas en cada caso. Por lo que pueden ser difíciles de mantener y actualizar a medida que cambian los requisitos de la tarea.

2.1.2. Modelos basados en aprendizaje automático

Estos modelos utilizan técnicas de aprendizaje automático para aprender a realizar tareas específicas a partir de los datos, sin necesidad de ser programados explícitamente para realizar dichas tareas. Algunos ejemplos populares de este tipo de modelos incluyen redes neuronales como la de la Figura 5 [1] y árboles de decisión [36].

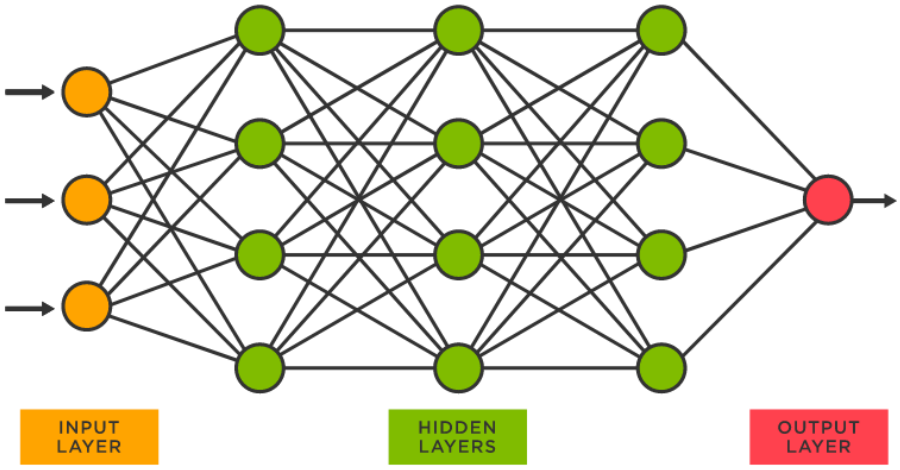

Figura 5: Modelo basado en aprendizaje automático.

Uno de los principales beneficios de los modelos basados en aprendizaje automático es su capacidad para aprender por sí mismos a partir de los datos. Esto significa, que pueden mejorar su rendimiento a medida que procesan más datos. Pueden ser más flexibles que los modelos basados en reglas, que dependen completamente de conjuntos de reglas predefinidas.

Existen varias categorías dentro del aprendizaje automático, dentro de las cuales se encuentra el aprendizaje supervisado y el no supervisado [44]. En el aprendizaje supervisado, los modelos aprenden a hacer predicciones a partir de ejemplos de entrenamiento etiquetados, donde se conocen tanto las entradas como las salidas. Su objetivo es aprender a hacer predicciones precisas sobre nuevos datos en base a los datos de entrenamiento. En el aprendizaje supervisado, los modelos aprenden a descubrir patrones o estructuras de los datos sin etiquetas.

Aunque los modelos basados en aprendizaje automático sean un gran acierto para el tratamiento de datos, estos también tienen algunos inconvenientes. En primer lugar, necesitan acceso a conjuntos de datos para entrenar, lo que puede ser costoso y laborioso de obtener. Además, pueden ser más lentos que los modelos basados en reglas y pueden ser más difíciles de interpretar y comprender.

En general, los modelos basados en aprendizaje automático pueden ser una buena opción para tareas complejas y para las que hay suficientes datos disponibles.

2.1.3. Modelos basados en lenguaje natural

Los modelos basados en lenguaje natural son capaces de procesar el lenguaje de una manera más real y precisa, utilizando técnicas de NLP (*Natural Language Processing*).

"Tratar computacionalmente una lengua implica un proceso de modelización matemática. Los ordenadores solo entienden de bytes y dígitos." (Moreno, 2022) [34]. Estos modelos permiten, por tanto, procesar y comprender textos. Por lo que, en algunos casos, pueden superar a los modelos basados en aprendizaje automático.

2.1.4. Modelo basado en CLIP

Una vez observados los diferentes modelos, se puede explicar por qué se ha elegido la herramienta CLIP para este proyecto.

La llegada de las redes neuronales como CLIP ha permitido alcanzar los objetivos propuestos y trabajar esta gran masa de información eficientemente. A una velocidad mayor que otros métodos tradicionales como es el caso de modelos basados en reglas. Ya que estos últimos son más lentos al no contemplan un enfoque general de los datos, y tienen que cambiar el algoritmo para cada tarea específica.

El sistema encargado para el procesamiento de datos no estructurados requiere estar conformado por componentes que puedan ser evaluados con las mismas características. Por ende, el procesamiento de datos comienza con la representación de estos.

Para poder trabajar con un sistema multimodal, es necesario habilitar el tratamiento de ambas informaciones dentro del mismo dominio. El sistema debe obtener una representación única y general para todos los tipos de datos no estructurados, en este caso imagen y texto. Además, esta debe ser robusta para cada elemento no estructural.

Las computadoras no entienden texto o imágenes como un ser humano. Solo entienden números, por lo que se precisa convertir los datos no estructurados en alguna representación numérica útil. De esta manera, los datos serán representados mediante *embedding vectors*.

CLIP implementa las funcionalidades de los modelos basados en aprendizaje automático no supervisado con las de los modelos basados en lenguaje natural. "CLIP (*Contrastive Language–Image Pre-training*) se basa en un modelo de transferencia *zero-shot*, supervisión del lenguaje natural y aprendizaje multimodal." (CLIP: Connecting Text and Images., 2021) [11].

La red neuronal CLIP ha sido creada por la empresa de OpenAI. Está basada en el aprendizaje por contraste que puede observarse en la Figura 6. Combina el procesamiento de lenguaje natural con el procesamiento de imágenes. Por lo que es un modelo que puede procesar elementos de entrada de múltiples modalidades. En el pre-entrenamiento, CLIP entrena al codificador de imágenes y al codificador de texto para predecir qué imágenes se emparejan con qué textos en el conjunto de datos.

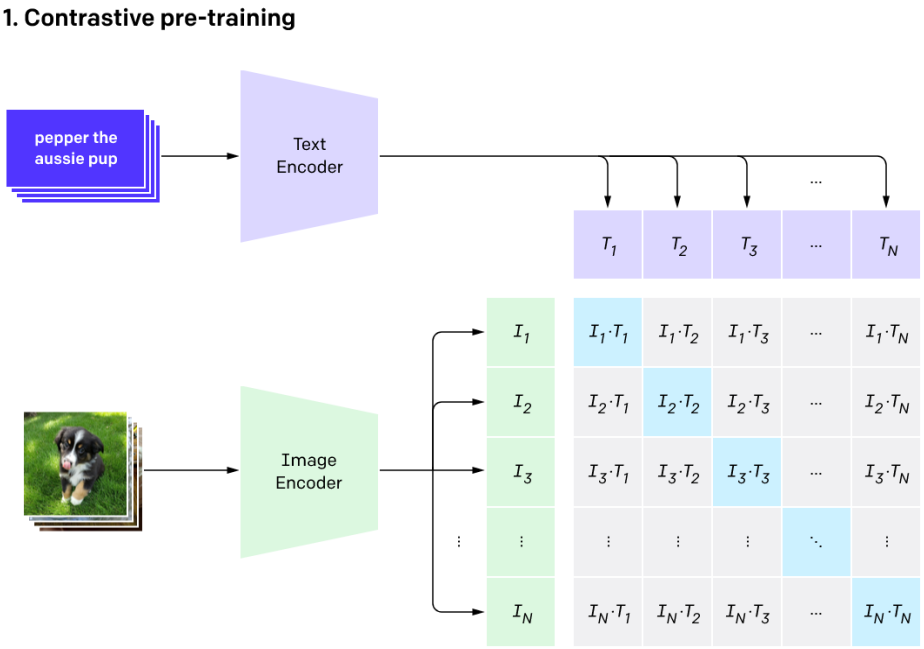

Figura 6: Pre-entrenamiento contrastivo.

El objetivo de CLIP es aprender una representación continua y generalizada de los datos de entrada que pueda ser útil para una variedad de tareas relacionadas con el lenguaje y las imágenes. En consecuencia, ha demostrado tener un rendimiento excepcional en una amplia variedad de tareas de procesamiento de lenguaje natural y visión por computadora.

CLIP genera vectores que describen y recogen las características esenciales de cada elemento a procesar. A su vez, permite evaluar en un mismo espacio

vectorial representaciones vectoriales de texto y de imagen. El modelo procesa la información para convertirla en *embedding vectors*.

CLIP aprende a asociar una amplia variación de conceptos visuales con descripciones de texto, es decir, empareja fragmentos de texto con imágenes y compara la similitud entre vectores.

> La red puede recibir instrucciones en lenguaje natural para realizar una gran variedad de puntos de referencia de clasificación, sin optimizar directamente el rendimiento del punto de referencia. Al no optimizar directamente para el punto de referencia, mostramos que se vuelve mucho más representativo. (CLIP: Connecting Text and Images., 2021) [11].

Anteriormente, los modelos de visión tradicional entrenaban conjuntos de datos etiquetados manualmente, identificando el contenido semántico y documentándolo, lo que requería mucho trabajo y esfuerzo. Este proceso sólo permitía supervisar una cantidad limitada de conceptos visuales. Estos modelos realizaban con precisión una única tarea, sin embargo, si se deseaban añadir otras, era obligatorio crear otro conjunto de datos, agregar sus etiquetas y ajustar el modelo para su buen desempeño.

En contraste, CLIP permite reducir la necesidad de grandes y costosos conjuntos de datos etiquetados. Ya que, mediante la generalización y transferencia del modelo, se pueden predecir categorías de objetos visibles o invisibles, utilizadas o no en el conjunto de entrenamiento general. CLIP puede realizar una amplia variedad de tareas de clasificación visual sin necesidad de ejemplos adicionales de entrenamiento ni de reajustar el modelo.

El aprendizaje *zero-shot* permite tratar los archivos multimedia sin tener que realizar un entrenamiento específico, sin optimizar los puntos de referencia ganando robustez y representación. Esto permite evaluar en puntos de referencia sin tener que entrenar con sus datos. La red está entrenada en una amplia variedad de imágenes con una extensa diversidad de supervisión de lenguaje natural que está abundantemente disponible en Internet. Por tanto, una vez pre-entrenado el modelo, CLIP estima qué imagen se empareja mejor con un texto dado, o viceversa.

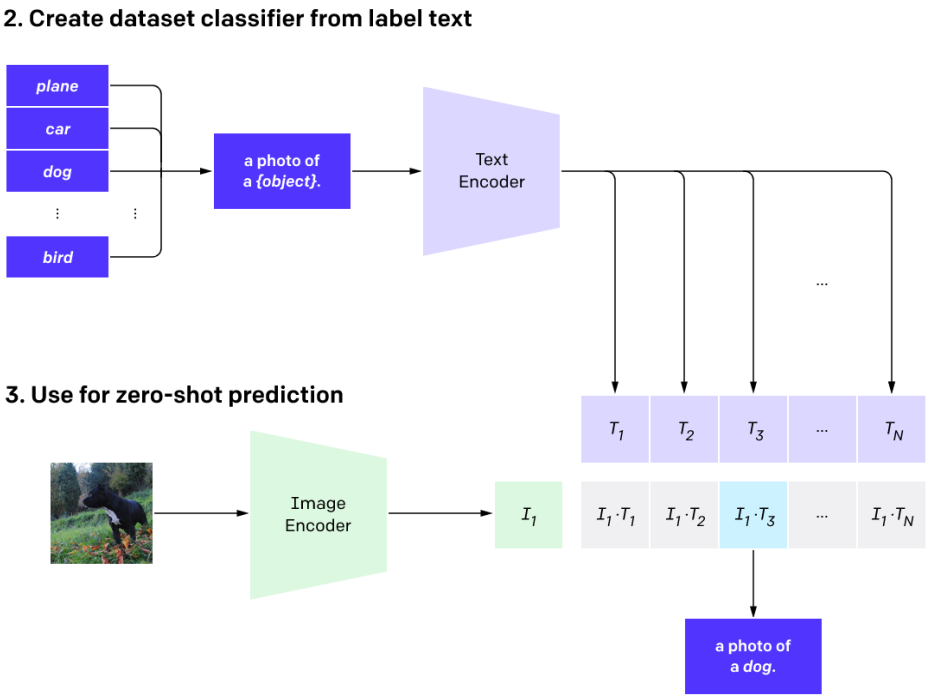

Figura 7: Aprendizaje zero-shot.

El modelo CLIP consigue aprender conceptos visuales con un entrenamiento contrastivo a partir de pares texto-imagen, de manera eficiente, flexible y general, haciendo frente a los principales problemas en los modelos de aprendizaje tradicionales y seleccionando las ventajas de los modelos más recientes.

2.2. Extracción de metadatos

La siguiente tarea por evaluar, plantea la idea de cómo convertir las imágenes y textos implicados en el objetivo final del proyecto, a un mismo dominio. Para que el proceso de comparación entre elementos multimodales sea posible.

Para la obtención de los elementos multimedia, se hace uso de la librería OpenCV (*Open Source Computer Vision Library*). Es una librería software de código abierto basada en visión artificial multiplataforma, que proporciona una infraestructura para aplicaciones de aprendizaje automático y visión por computadora [2]. Está compuesta por una gran cantidad de algoritmos universales, que han sido optimizados, para el procesamiento de imágenes. Permiten realizar un gran número de tareas. OpenCV junto con la herramienta de Python permitirá capturar cuadros de video a intervalos específicos, y guardarlos como imágenes para su posterior modelado en vectores. Los logos de estas herramientas pueden verse en la Figura 8.

Figura 8: Herramientas utilizadas para la extracción de metadatos.

Gracias a esta librería, se puede obtener información característica acerca de un vídeo dado. Tales como la duración total del archivo especificado, la velocidad a la cual aparece cada fotograma, el tiempo exacto en el que se encuentra un fotograma concreto dentro del conjunto.

Al capturar cada fotograma, se hará uso del modelo de red neuronal CLIP para obtener los *embedding vectors*. En la Figura 9 [39], puede observarse un ejemplo de cómo una imagen se transforma en un *embedding vector*. Los vectores serán enviados al proceso de segmentación junto a los demás metadatos extraídos mediante OpenCV.

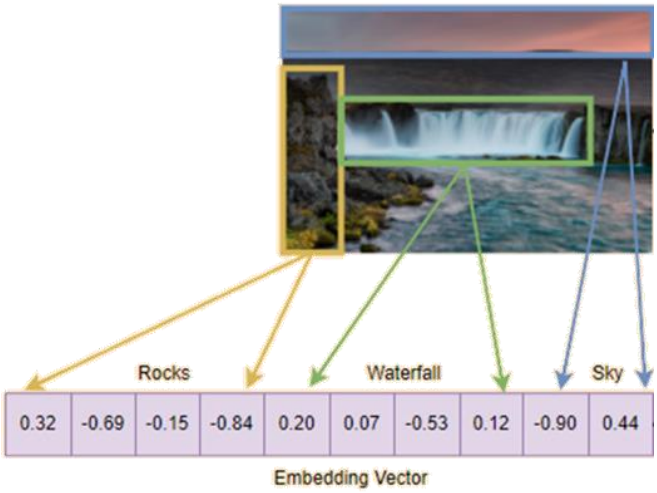

Figura 9: Representación de una imagen en un embedding vector.

Gracias a la introducción del entrenamiento *zero-shot* de CLIP, se ha conseguido predecir la clasificación de objetos sin necesidad de modificar los modelos ya pre-entrenados. Esto supone una mejora al incluir la posibilidad de no necesitar disponer de un conjunto de entrenamiento inmenso y concreto.

Esta red no supervisada permite a la vez, ordenar el contenido capturado de diversas maneras. Incluyendo un índice que pueda identificarlo, y extrayendo metadatos útiles que ayuden a efectuar búsquedas de objetos, textos, rostros o incluso contextos. Además de permitir diferentes modalidades de entrada, ya sea texto o imágenes.

2.3. Métodos de segmentación

Reducir el tamaño de los archivos en el mundo multimedia se ha convertido en una necesidad muy presente. Encontrar un método que permita fragmentar el contenido para evaluar qué información y cuál no es relevante, es decir, es de mayor o menor interés. O si existiera información repetida, cómo prescindir de ella.

El nivel más elemental en el que se puede representar un video es el fotograma. Generalmente la extracción del conjunto de fotogramas permite el análisis de las características visuales como color, textura o forma de las imágenes. Los fotogramas permiten analizar el siguiente nivel, la representación por las tomas.

"Una toma es el conjunto consecutivo de imágenes capturado por una cámara entre las operaciones de inicio y fin de la grabación, la cual marca los límites de la toma." (Hu, Xie, Li, Zeng, & Maybank, 2011) [18]. En ella se incluyen movimientos de cámara, así como movimientos de objetos pequeños. En este trabajo se ha estudiado la definición de tomas, identificando las transiciones que las separan, para finalmente evaluar en un futuro la generación de secuencias mediante la agrupación de tomas.

Existen diferentes mecanismos de segmentación, los cuales aparecen en el de artículo *A review of real-time segmentation of uncompressed video sequences for content-based search and retrieval.* (Lefèvre, Holler, & Vincent, 2003) [26]. Según el modo y las características sobre las que se desea realizar el análisis, entre los métodos más conocidos destacan:

2.3.1. Segmentación manual

La descripción como la indexación del contenido deben representar las distintas partes del video que constituyen por sí solas un único significado propio.

Segmentar un vídeo, determinando cuáles son los fragmentos que constituyen escenas, es una tarea compleja, ya que se trata de un proceso de selección en el que no existe una estandarización predefinida. Los diferentes formatos en los que se presentan el contenido hacen que cada profesional pueda analizar subjetivamente una estructura u otra. Por lo que realizar esta tarea manualmente resulta costosa tanto en tiempo como en esfuerzo.

2.3.2. Segmentación automática

La segmentación automática consiste en la división o extracción de partes de un vídeo digital mediante un algoritmo. Estas técnicas permiten esencialmente trabajar con características visuales. La implementación de sistemas de segmentación basados en el análisis visual para calcular su similitud es una tarea

difícil. La idea principal de estas técnicas es agrupar aquellas imágenes que comparten atributos similares. Según qué característica se quiera analizar, se determinará cómo será la detección de los límites de una toma.

En este apartado se ha seguido la diferenciación de tipos de segmentación automática seguida por el artículo, *Segmentación de vídeos informativos en televisión: de la práctica profesional a la identificación automática* (Caldera-Serrano & Caro-Castro, 2018) [8]. Junto a la descripción de los métodos de segmentación automática que ofrece el artículo, *Implementación de Interfaz Gráfica para Comparación Visual de Métodos de Segmentación y Procesamiento de Video, Usando Matlab* (Lascano, Pombo, & Chavez-Burbano, 2011) [25].

Se distinguen los siguientes métodos de segmentación automática:

2.3.2.1. Segmentación por color

Los algoritmos de segmentación pueden utilizar diferentes métodos para analizar el color en cada fotograma. Mediante este análisis, se puede deducir que pertenecen a la misma toma, aquellos fotogramas consecutivos en los que los colores que los componen sigan constantes.

Existen dos fórmulas para realizar la segmentación de video por color, la primera de ellas implica la detección de un cambio debido a la ausencia de colores existentes en fotogramas anteriores. Apagando un color y manteniendo encendidos el resto. Y la segunda, implica la detección de un cambio debido a la aparición de nuevos colores inexistentes en fotogramas anteriores, esto es, encendiendo un color y apagando el resto.

Debido a la amplia gama de colores disponible en las imágenes multicanal y teniendo en cuenta que el tiempo de procesamiento de video se hace más pesado por cada color que uno desee ingresar a la segmentación, normalmente se trabaja con los colores primarios RGB, rojo, verde y azul.

Entre las técnicas más conocidas utilizadas en la segmentación por color, se encuentra la segmentación por umbralado. Consiste en asignar un umbral a cada canal de color de la imagen, o a la escala de grises si se decide analizar los fotogramas en tonos entre blanco y negro. En la Figura 10 se observa un ejemplo de segmentación mediante un umbral T [32].

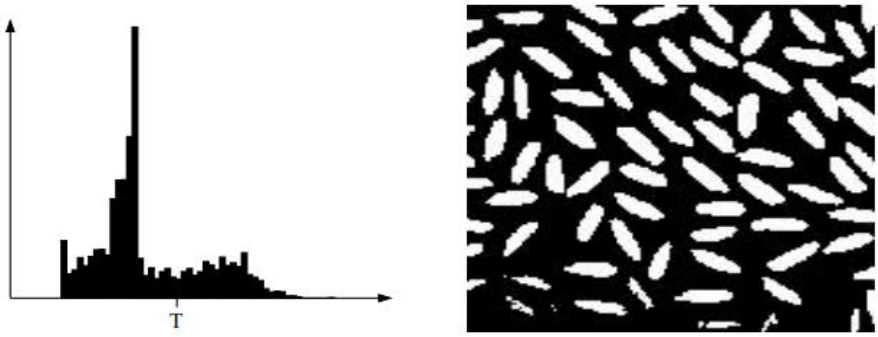

Figura 10: Método de umbralización.

A continuación, se analiza el valor en ese canal para cada píxel. Cuando dicho valor sea menor al umbral, será considerado como fondo, mientras que los píxeles con valores mayores serán considerados de interés. Se identificarán con los objetos predominantes en la escena de video.

Para evaluar la relación entre fotogramas adyacentes para identificar lo que está sucediendo en el vídeo, se estudia la evolución de esos píxeles de interés a lo largo del conjunto de imágenes. Cuando estos presentan un cambio, se detecta una toma nueva. Es decir, pasan de superar el umbral a no superarlo, siendo ese instante el marcado como cambio de toma.

El histograma de color es otra de las técnicas más populares dentro de la segmentación por color. Un histograma representa gráficamente una distribución de frecuencias. En el caso de una imagen, representa las frecuencias de los diferentes valores de color en la imagen. Se puede obtener un histograma por cada color primario o histogramas conjuntos que aportan información sobre el rango de colores más frecuente.

La idea que sostiene este método es identificar si los histogramas entre fotogramas consecutivos han sufrido un cambio o si son semejantes las distribuciones, mediante la intersección de estos.

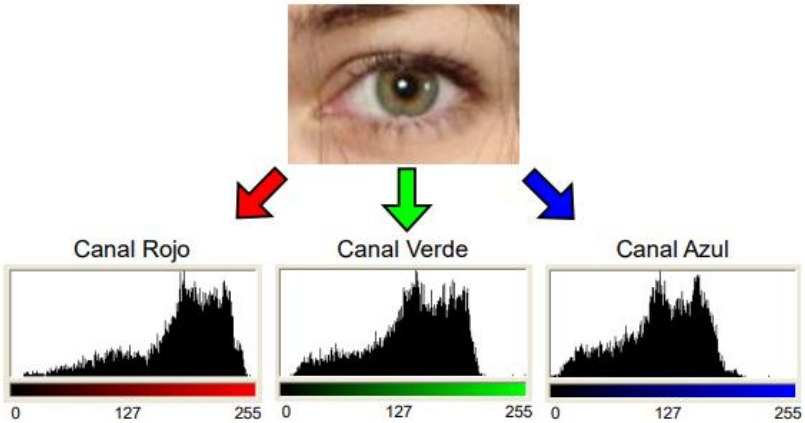

Figura 11: Histograma de color.

Permite diferenciar cambios de escena debido a que son sensibles frente a cambios grandes de cámara, donde las diferencias de color suelen ser notables. Aunque por contra, debido a su robustez frente a cambios pequeños de cámara ya que en este tipo de movimientos no hay un cambio significativo de color, no pueden diferenciar claramente los cambios de toma dentro de una misma escena.

Otro algoritmo similar al anterior es el modelo de mezcla de gaussianas. "Se basa en la aproximación de la distribución de colores generada por un dispositivo utilizando una mezcla de gaussianas en el espacio RGB." (Sánchez & Binefa, 2000) [38]. Dichas representaciones se comparan entre sí para la identificación de cambios de color dentro del video, por ejemplo, usando el algoritmo E-M (*Expectation-Maximization*) [23].

2.3.2.2. Segmentación por formas

Estas técnicas permiten dividir un video en diferentes segmentos basándose en la forma de los objetos en cada fotograma. Esto se logra mediante el análisis de las características geométricas de los objetos como el análisis de contornos, la detección de objetos o regiones y el análisis de movimiento.

Uno de los métodos utilizados es la detección de valores de intensidad de las zonas. El algoritmo suaviza la imagen reduciendo el ruido producido por los detalles y las texturas de los objetos, y posteriormente, transforma cada fotograma dentro de una escala de grises para que los colores no puedan confundirse con los bordes. De esta manera los contornos quedan bien definidos. Se identifican los píxeles en los que se produce máxima variación, eliminando aquellos que varían de forma poco significativa. Una vez detectados los bordes de los objetos, se analiza el progreso de estas formas que se dibujan para evaluar la existencia de un cambio de escena o no.

Figura 12: Detección de bordes.

Los algoritmos basados en detección de formas pueden ser combinados con otros métodos como la segmentación por colores. "En una primera fase, se segmenta buscando áreas homogéneas de similar cromaticidad. Y, en segundo lugar, se detectan los contornos de las áreas segmentadas y se construyen los bordes para esas regiones detectadas." (Gil, Torres, & Ortiz Zamora, 2004) [17].

El uso de métodos combinados, refuerzan la detección de los objetos. Por ejemplo, para evaluar qué está en primer plano y qué se encuentran en el fondo. Para finalmente, identificar el movimiento de los objetos mediante vectores de movimiento. Este procedimiento permite identificar cuando se ha producido un movimiento ligero en la escena, y cuando se ha producido un cambio importante, los fotogramas contiguos no tendrán gran relación, por lo que identificarán un cambio de plano.

La segmentación por formas es válida en videos que presentan cambios de objetos importantes en las tomas, si los objetos que aparecen en escena tienen formas parecidas podrían confundirse.

2.3.2.3. Segmentación por textura

Estos métodos permiten segmentar un video basándose en la apariencia visual de la textura de los objetos de la imagen. Por textura se hace referencia a las diversas regiones de una imagen cuyo contenido puede definirse midiendo la luminosidad, la aspereza, la suavidad, los brillos y sombras, la rugosidad, la existencia de huecos, entre otras características.

La identificación de texturas se logra mediante el análisis de patrones dentro de una región de la imagen, es decir, reconocer áreas específicas que se basan en una misma textura. Un ejemplo es el caso de la Figura 13 [41], donde separa la imagen en *superpíxeles*, formados por conjuntos de píxeles que tienen la misma textura.

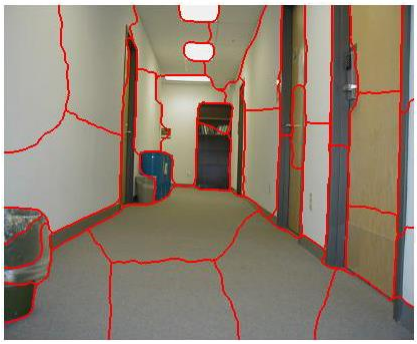

Figura 13: Detección de texturas.

En cada píxel se analiza la intensidad de la luz en comparación con sus adyacentes, para asignar regiones a la imagen que presentan la misma textura. Cuando estas texturas cambian de forma significativa entre fotogramas, se lleva a cabo la segmentación de video.

Al igual que la segmentación por formas, es de gran utilidad transformar las imágenes a una escala de grises para que luego, los fotogramas atraviesen un filtro o máscara, que identifica la textura.

2.3.2.4. Segmentación semántica

La implementación de un modelo de segmentación semántica juega un papel central en el reconocimiento de objetos y contextos de escenas coherentes. La utilización de la información visual desde un punto de vista de significado aporta un gran valor en el proceso de segmentación, ya que no sólo consigue diferenciar planos, sino que también contextualiza el contenido de la escena.

El incentivo es crear un método que indique entre qué fotogramas existe relación y en qué momento hay un cambio significativo entre fotogramas. Es decir, en qué instante del video dos fotogramas consecutivos no tienen relación alguna, son independientes, pertenecen a contextos distintos.

La segmentación semántica pretende comprender el contenido del video e identificar si la relación entre fotogramas adyacentes tiene un significado global. Asigna etiquetas a cada conjunto de fotogramas de manera automática, sin hacer uso de un profesional que mediante juicios subjetivos realice la partición manualmente. Este tipo de fragmentación permite hacer separaciones más naturales y realistas desde un punto de vista humano. Aunque también resulta ser una segmentación que depende del punto de vista que se tome.

En este proyecto, se ha optado por usar una segmentación temporal por detección de cambio de plano, clasificando las secuencias de fotogramas en tomas según su semántica. A medida del paso de fotogramas se identifica si existe una alteración respecto a la imagen anterior.

En el proyecto se va a conseguir esta segmentación semántica mediante el cálculo de una matriz la cual refleja la distancia coseno de un vector con el resto. El algoritmo implementado va comparando vectores uno a uno y calculando su distancia entre ellos. Es decir, la similitud que existe entre ambos *embedding vectors*.

La Figura 14, presenta un diagrama de color, con los cálculos de dicha matriz. El valor 1 indica que los vectores son idénticos, mientras que el valor 0, indica que son completamente opuestos. De esta manera, la representación de esta matriz, indica los momentos en los que fotogramas consecutivos, presentan una diferenciación. La matriz será cuadrada de dimensiones: número de fotogramas total por número de fotogramas total. Puede observarse la existencia de grupos de *frames* similares entre sí, representados por bloques de color amarillo.

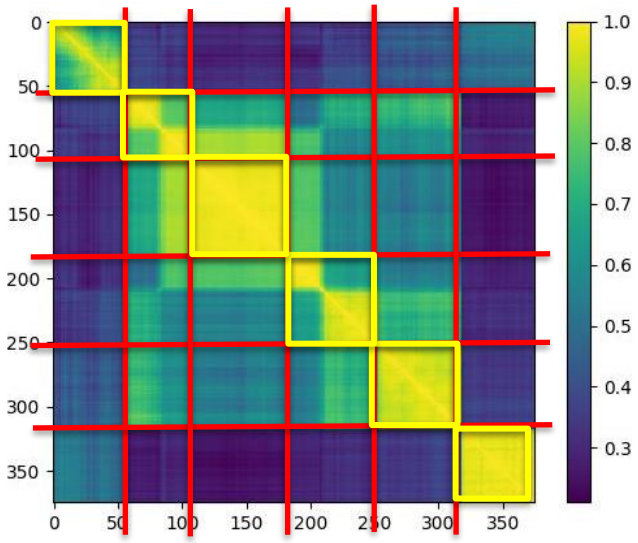

Figura 14: Segmentación semántica.

Para automatizar la división de tomas, se ha realizado el cálculo del gradiente de la matriz. Siendo esta simétrica, basta con el cálculo respecto a uno de los ejes. En la Figura 15, se puede ver el cálculo del gradiente de la matriz mencionada anteriormente. Para determinar la limitación entre tomas, se ha utilizado el método de umbralización.

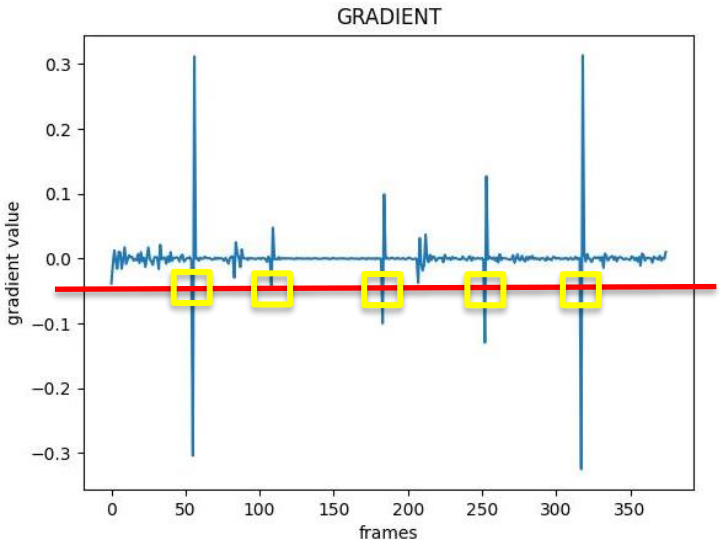

Figura 15: Cálculo del gradiente.

Capítulo 3

Base de datos Milvus

En este capítulo se va a hablar de la posibilidad que ofrece Milvus para hacer frente a la necesidad de sistemas de almacenamiento, indexado y búsqueda dentro del estudio de la industria multimedia y el porqué del uso de esta tecnología.

3.1. Búsqueda y Recuperación de Información

> La búsqueda y recuperación de información es una tarea compleja, que depende mucho de la capacidad, los conocimientos y los intereses de la persona que la ejecuta y que puede ser un fracaso, si no se tiene precisión de la necesidad informativa [9].

Es el proceso de encontrar y recopilar información relevante a una determinada consulta o pregunta. Esto se consigue utilizando diversas técnicas y herramientas, como motores de búsqueda, bases de datos, bibliotecas y recursos en línea.

La idea del uso de computadoras para la búsqueda de fragmentos relevantes de información se popularizó a raíz del artículo *As We May Think.* (Bush, 1945) [7].

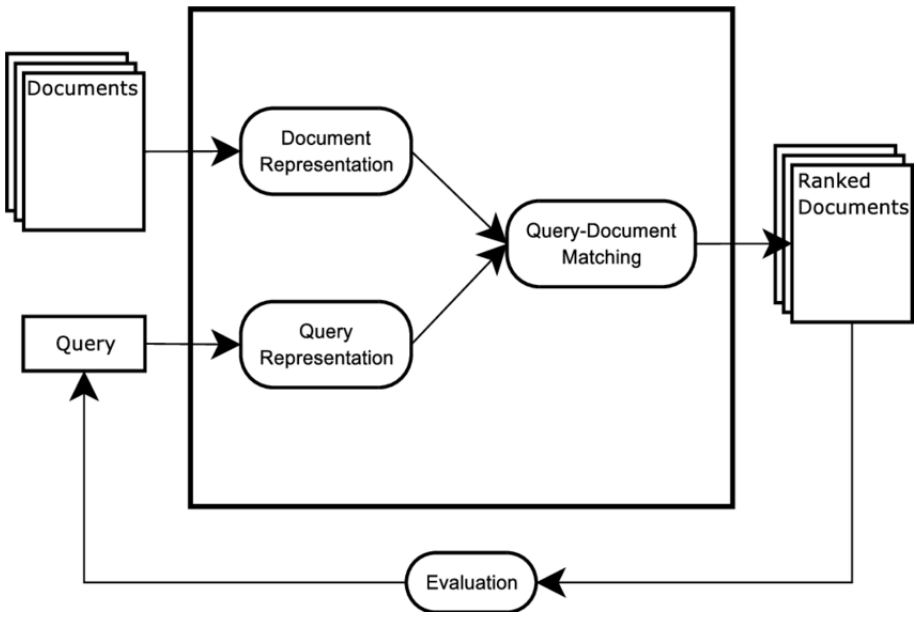

Figura 16: Sistema de recuperación de información.

La ISR (*Information Search and Retrieval*), es la ciencia de la búsqueda de información tanto en cualquier tipo de documentos electrónicos y digitales como en el interior de estos mismos. En la Figura 16 [5], se puede observar un esquema que explica lo que es un sistema de recuperación de información.

Entre las bases de la ISR se encuentra la búsqueda de metadatos que describan los documentos, la realización de búsquedas en bases de datos, entre otras. Y como objetivo, lleva a cabo la recuperación de información en textos, imágenes, audios o datos de otras características de manera pertinente y relevante.

Los sistemas de ISR utilizan técnicas de procesamiento de señal, para recopilar información de diferentes fuentes, y luego analizarla extrayendo información útil.

Las múltiples interpretaciones y aplicaciones de los diferentes tipos de datos multimedia impiden la creación de un estándar sobre indexación y mecanismos de búsqueda. Por lo que, para alcanzar el objetivo de recuperación, es imprescindible emplear los sistemas de información en conjunto con la informática. Y fijar unos criterios de búsqueda que permitan realizar consultas en los datos de forma eficaz, coherente y sencilla. La ISR constituye una rama de la informática enfocada en el desarrollo de sistemas de recopilación de información.

Siendo la multimedia un tipo de información bastante complejo, hay que tener en cuenta que la recuperación de información audiovisual debe basarse principalmente en el contenido. Dentro del estudio de archivos de video, las recuperaciones de información incluyen el análisis y el modelo del contenido, la extracción de características, la indexación y por último la búsqueda de elementos.

3.2. Datos no estructurados

Antes de realizar un análisis de los datos, es esencial identificar qué tipo de contenido se va a almacenar, para saber cómo deben ser tratados los datos. La administración de datos no estructurados es uno de los mayores problemas que se presentan en la actualidad. Esto se debe a que las herramientas y técnicas existentes, han sido desarrolladas para el tratamiento de información estructurada. “Estamos acostumbrados a los datos estructurados, esto es, aquellos que están basados en “campos”. Y a la hora de procesar datos no estructurados fallan.” (Macarrón, Tipos de Datos no relacionales., 2021) [29].

Sin embargo, en los últimos años, la industria de la información cada vez se enfoca más en el uso de archivos y documentos no estructurados como el material multimedia. “Los datos no estructurados suponen un 80% del volumen de todos los datos generados, y el porcentaje no deja de crecer.” (Tech, 2023) [43]. Por lo que se ha tomado conciencia de la necesidad de crear sistemas que permitan organizar, almacenar y procesar este tipo de información. Y que, de esta manera, se consiga

aumentar la flexibilidad de los sistemas extrayendo información significativa para que los datos sean más accesibles.

Para ello, se disponen de métodos de segmentación que permiten agrupar la gran cantidad de datos, condensando lo más representativo del conjunto y prescindiendo de información demasiado detallada. Los procesos de segmentación son esenciales para realizar búsquedas en entornos no estructurados, ya que permiten condensar la información más relevante.

La información no estructurada presenta particularidades respecto a la información sí estructurada, debido a que no se siguen unas normas predefinidas para su administración y manipulación. Es decir, no tiene un modelo formal de dato que la represente o en caso de tener uno, no es fácilmente utilizable por un programa de computadora.

En el caso que concierne, el contenido que aparece en los archivos de video dentro del mundo audiovisual está categorizado como datos no estructurados. La información se encuentra implícita desde el punto de vista humano, es decir, puede obtenerse a partir de simple observación, visualización de los fotogramas, contextualización, lectura... Pero no puede ser interpretada por computación. Los datos multimedia están representados por una colección de características que hay que explotar.

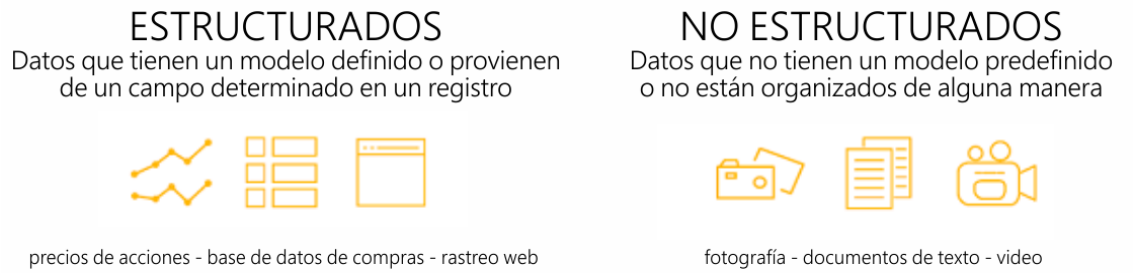

Figura 17: Tipos de datos.

Lo que se pretende es crear un sistema que pueda extraer automáticamente la información contenida en estos documentos utilizando las estructuras lingüísticas y visuales inherentes, para que así sea posible su interpretación por computadora. El sistema por implementar debe ser capaz de tratar rápidamente gran cantidad de datos muy variados entre sí y de diferente naturaleza, y a su vez, realizar tareas de procesamiento de lenguaje natural y visión por computadora.

3.3. Bases de datos relacionales y no relacionales

Una de las principales ideas que se ha tenido en cuenta es cómo guardar los datos. Para ello, se han evaluado las características que presentan cada tipo de base de almacenamiento, para escoger la adecuada para el tipo de datos que se va a

procesar. Se ha consultado para ello, el artículo *¿Qué diferencia una base de datos relacional vs una no relacional?* (Arregui, 2022) [4].

La primera labor es encontrar un registro que permita almacenar gran cantidad de datos no estructurados, además de información extra que sea del interés del usuario. Que pueda utilizarse sin la necesidad de hacer modificaciones en la estructura entre archivos de video con características completamente diferentes.

Por tanto, será de interés utilizar una base de datos que permita recoger cierta información específica para luego realizar operaciones y estudios que aporten valor útil en su investigación. Viendo los requisitos y fundamentos del proyecto, será necesario la utilización de una base de datos principalmente no relacional.

3.3.1. Base de datos relacional

Las arquitecturas de datos relacionales almacenan la información en forma de tablas con estructuras fijas de filas y columnas, como puede observarse en la Figura 17 (Macarrón, Tipos de bases de datos no relacionales., 2021). Cada tabla tiene una clave principal que se utiliza para establecer relaciones entre tablas. Estas relaciones permiten que los datos se dividan en tablas lógicas y se relacionen entre sí mediante consultas SQL. Las bases de datos relacionales son mejores para aplicaciones con un esquema fijo y un conjunto de reglas de integridad de datos. Tratan principalmente con datos estructurados, siguiendo patrones definidos.

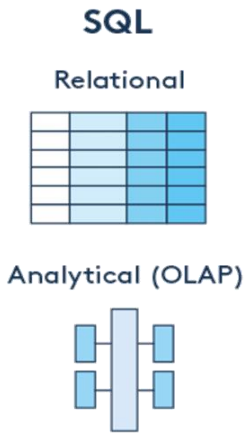

Figura 18: Bases de datos SQL.

Son muy intuitivas de implementar, por lo que, a diferencia de otras, son más sencillas de utilizar. Permiten la uniformidad de los datos y evitan la duplicidad de los registros, favoreciendo más la comprensión y la accesibilidad. Además, son bases de datos robustas y con menos vulnerabilidades ante fallos, ya que garantizan consistencia e integridad.

Por contra, como nombra uno de los artículos consultados durante el proyecto [46], son difíciles de mantener en el momento que el volumen de datos

aumenta. Este factor se debe también a que este tipo de base de datos presenta un espacio limitado de almacenamiento. Lo que puede provocar problemas graves si se van a guardar grandes cantidades de archivos. Y, por último, ha de decirse que, aunque sí tienen un buen rendimiento, este no aplica en caso de hacer consultas y obtener información, debido a la estructura de tablas separadas.

3.3.2. Base de datos no relacional

Aunque resulte llamativa la utilización de una base de datos relacional, en este trabajo interesa utilizar una base de datos no relacional.

> Hay diferentes situaciones y motivos por los que las bases de datos no relacionales son el entorno idóneo: por ejemplo, vídeos, audios, imágenes, texto "puro y duro", … Son algunos de los tipos de datos que no se "llevan bien" con un sistema relacional. (Macarrón, Azure y bases de datos no relacionales., 2021) [27].

Las bases de datos no relacionales son altamente flexibles debido a la variedad de tipos de datos que pueden almacenar, por lo que se ha optado por una arquitectura no relacional.

Las bases de datos NoSQL, están diseñadas para soportar grandes volúmenes de datos, lo que permite su fácil escalabilidad a través de múltiples servidores. La información se suele almacenar en documentos, objetos, grafos o clave-valor como indica la Figura 18 [28]. Lo que permite gestionar y organizar información no estructurada, donde se desconoce la forma de los datos que se pretenden almacenar. Estas no cuentan con un sistema estandarizado, lo que aporta un cierto grado de libertad. Aunque, por el contrario, suelen ser algo menos sencillas de utilizar. Permiten ejecutar consultas de diferentes tipos de datos, son bases con el fin de lograr tareas variadas manejando metaconocimiento. Algunas de las más conocidas son MongoDB, Cassandra y Elasticsearch.

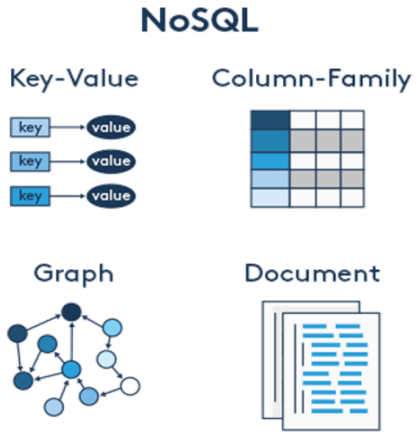

Figura 19: Bases de datos NoSQL.

3.3.3. Base de datos Milvus

Observando las ventajas e inconvenientes de cada tipo de base de datos, se ha llegado a la conclusión de utilizar la base de Milvus, ya que se trata de una base de datos no relacional que incluye varios beneficios de las arquitecturas relacionales.

Milvus es una base de datos vectorial nativa de la nube y de código abierto para aplicaciones de inteligencia artificial. “Milvus se creó en 2019 con un único objetivo: almacenar, indexar y administrar *embedding vectors* masivos generados por redes neuronales profundas y otros modelos de aprendizaje automático (ML).” (What is Milvus vector database?, 2022) [47]. Se usa ampliamente en escenarios como visión por computadora, procesamiento de lenguaje natural, sistemas de búsqueda y más.

Milvus, por tanto, es un motor de búsqueda vectorial que ha sido diseñado para manejar datos no estructurados, como archivos multimedia, y datos relacionados con el lenguaje natural. Milvus requiere como entrada un conjunto de vectores, de ahí que se haya utilizado el modelo CLIP para transformar los datos.

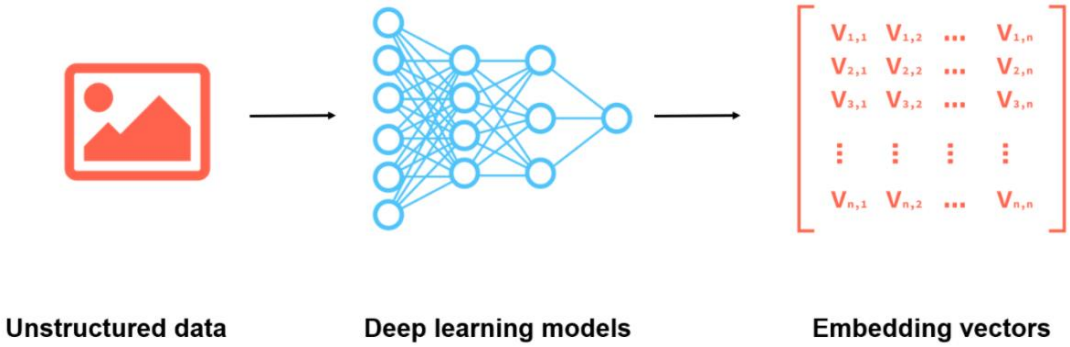

Figura 20: Extracción de embedding vectors a partir de datos no estructurados.

Milvus es considerada una base de datos no relacional, aunque almacena los datos en una estructura similar a una tabla, lo que permite su simplicidad, esta no tiene establecidas a priori las relaciones entre elementos.

Se va a utilizar la implementación Milvus standalone en la cual todas las operaciones, tanto la creación de índices como la búsqueda de similitud entre vectores, se completan en un solo proceso.

Para la instalación de Milvus standalone, se ha requerido la herramienta Docker Desktop cuyo logo aparece en la Figura 20 [14]. Instalado en Windows junto a WSL 2 [15]. Para la instalación de WSL se ha seguido la documentación de la empresa Microsoft [21]. Dicha instalación ha permitido el uso de la herramienta Docker Compose, para conectarse al servidor Milvus vía la dirección local del host y el puerto 1930. (Manage Milvus Connections., 2022).

Figura 21: Herramienta Docker.

Ejecuta tres contenedores Docker una vez abierto el programa, como se observa en la Figura 21 [30]: el servicio Milvus standalone, *milvus-standalone*, y sus dos dependencias *milvus-etcd* que almacena metadatos y *milvus-minio* que almacena objetos.

NAME	IMAGE	STATUS	PORT(S)
repo 33d1e8f1b11b	alpine/git:latest	Exited (128)	-
tfg 3 containers	-	Running (3/3)	-
milvus-standalone d9bd13882d64	milvusdb/milvus:v2.1.0	Running	19530
milvus-minio 281a602aed38	minio/minio:RELEASE.2022	Running	-
milvus-etcd 2586d997869c	quay.io/coreos/etcd:v3.5.0	Running	-

Figura 22: Contenedores Docker.

Permite crear colecciones [13] para almacenar y administrar las entidades. Cada colección está definida por un esquema el cual contiene el nombre de la colección, las características y el tipo de datos a almacenar en ella, además de una breve descripción.

El esquema debe contener un campo de clave principal y un campo vectorial. Adicionalmente se pueden implementar otros campos para almacenar parámetros relacionados. Los datos por insertar en la base de datos deben coincidir en formato con el esquema de la colección. Esta última característica aporta la sencillez de las bases de datos relacionales, ya que el formato de almacenamiento tiene ciertas analogías.

Se ha implementado un documento en Python que creará la estructura de las colecciones [31] que se va a seguir. Contiene en primer lugar un índice que indica la posición de la toma dentro del video, el cual se le ha llamado *video_id*, será el elemento primario del esquema. Este parámetro irá seguido por un vector, cuya longitud dependerá del modelo CLIP escogido, el 'RN50' [24], que transforma los fotogramas en vectores de dimensión 1024. A continuación, se dispondrá del rango de *frames* que componen la escena, *video_start* la imagen de comienzo, y *video_stop* la del final. Y por último el *string* que indica el *path* del video.

Se debe destacar que los propios *embedding vectors* presentan un índice [6] a parte de la palabra clave que se le ha asignado como identificador. Dependiendo del formato de los *embedding vectors*, en este caso *FLOAT_VECTOR*, Milvus admite un tipo de índice u otro.

Dentro de las categorías permitidas [20], se ha escogido la indexación mediante índices de tipo FLAT, que es el único que consigue precisión perfecta y una tasa de recuperación cerca del 100%. No comprime los vectores y garantiza resultados de búsqueda exactos. Esta precisión se debe a que FLAT requiere un enfoque exhaustivo, lo que significa que, para cada consulta, la entrada se compara con cada vector en el conjunto de datos. Resulta ser el índice más lento, pero compensa con el hecho de que no necesita entrenamiento de datos ni almacenamiento adicional.

Una vez determinada la estructura de la colección, se extraerán todos los fotogramas asignándoles a cada uno sus características. Los datos pasarán por el proceso de segmentación y tras la obtención de los segmentos se elaborará el proceso de indexado. En este proceso se implementará un índice a cada elemento a almacenar. Después, se podrá proseguir con la última fase del proyecto, la búsqueda.

Se usan métricas de distancia mediante *Approximate Nearest Neighbor* (ANN) algoritmos de búsqueda para calcular el parecido entre vectores. Si ambos son muy similares significa que están relacionados contextualmente. Milvus utiliza la búsqueda de similitud vectorial o búsqueda del vecino más cercano. En la consulta K-Vecinos, “se buscan los 'k' elementos más cercanos a un elemento 'q' dado.” (Salvetti, 2009) [37]. Es el proceso de calcular la similitud o distancias por pares entre un elemento de referencia 'q', siendo este el elemento consulta, y una colección de elementos existentes, aquellos que se encuentran dentro de la base de datos. Y devolviendo los 'k' vecinos más cercanos, que son los 'k' elementos más similares. Se ha seleccionado la búsqueda mediante el producto IP (*Inner Product*). Para evitar que Milvus realice búsquedas predeterminadas, se utilizan índices para organizar los datos.

En resumen, Milvus es un sistema de almacenamiento de vectores con características de búsqueda de similitud entre vectores, dentro de la colección de la base de datos seleccionada. Utiliza para ello un índice de vecinos cercanos para recuperar vectores similares a la consulta.

Capítulo 4

Pruebas y Resultados

Tras haber modelado los datos en *embedding vectors* (Anexo I.1) y extraído sus metadatos (Anexo I.2), se ha implementado el resto del código en Python que compone el sistema. Se han realizado una serie de pruebas acorde a los objetivos del proyecto. El código consultado para la realización del sistema de indexado y búsqueda viene del artículo *Video Search Engine Using OpenAI's CLIP* (Robinson, 2022) [35].

En este capítulo, se va a hablar de las técnicas y medidas empleadas dentro del sistema para la comprobación de su funcionamiento.

La elección de una medida adecuada para identificar tanto los límites de las tomas como la relación existente en la base de datos con las consultas, es un factor clave que determina el éxito o el fracaso de ambos procesos. Afectará a los resultados de la segmentación y al porcentaje de acierto en la realización de búsquedas.

4.1. Medidas de similitud

Para calcular la similitud entre dos elementos, es necesario que ambos estén representados dentro del mismo dominio. Como se ha comentado anteriormente, el proceso de segmentación que se ha llevado a cabo es del tipo de segmentación semántica (Anexo I.3).

Una de las medidas más conocidas para el cálculo de la similitud entre los vectores que conforman los fotogramas es la similitud basada en coseno. "En este caso, los elementos se consideran dos vectores en el espacio de usuario dimensional. La similitud entre ellos se mide calculando el coseno del ángulo formado por la intersección de las representaciones vectoriales." (Mendoza Olguín, Laureano De Jesús, & Pérez de Celis Herrero, 2019) [33].

El valor 1 indica que los vectores son parecidos, es decir, tienen la misma orientación. Mientras que el valor 0, indica que son diferentes, entre sí están orientados 90º. "Los vectores unitarios cumplen con la condición de ser máximamente "similares" si son paralelos y en el caso de que sean máximamente diferentes, deben ser ortogonales o perpendiculares entre sí." (Algoritmo de similitud de coseno., 2015) [3].

Se ha utilizado la función *distance.cosine*, de la librería Scipy, usando como entrada dos *embedding vectors*. Dicha función implementa la siguiente ecuación:

$$D(u,v) = 1 - \frac{u \cdot v}{\|u\|_2 \cdot \|v\|_2} \quad (1)$$

La similitud se define entonces como:

$$S(u,v) = 1 - D(u,v) \quad (2)$$

Se ha calculado una matriz de distancias coseno cuyo elemento (i,j) corresponde con la comparación del vector i-ésimo con el vector j-ésimo.

Para reducir costes de computación, se ha decidido por calcular la mitad de las comparaciones. Esto es, los cálculos son conmutativos, existe la misma distancia entre el vector i-ésimo y el vector j-ésimo, que entre el vector j-ésimo y el vector i-ésimo. Se ha realizado el computo de la matriz triangular superior y se ha replicado simétricamente a la triangular inferior. En la diagonal se encontrará la similitud máxima ya que se estará comparando un vector consigo mismo. En la Figura 23, se ha realizado una representado la matriz de distancias coseno de un video de 5 minutos *'video_bbc_01.mp4'*.

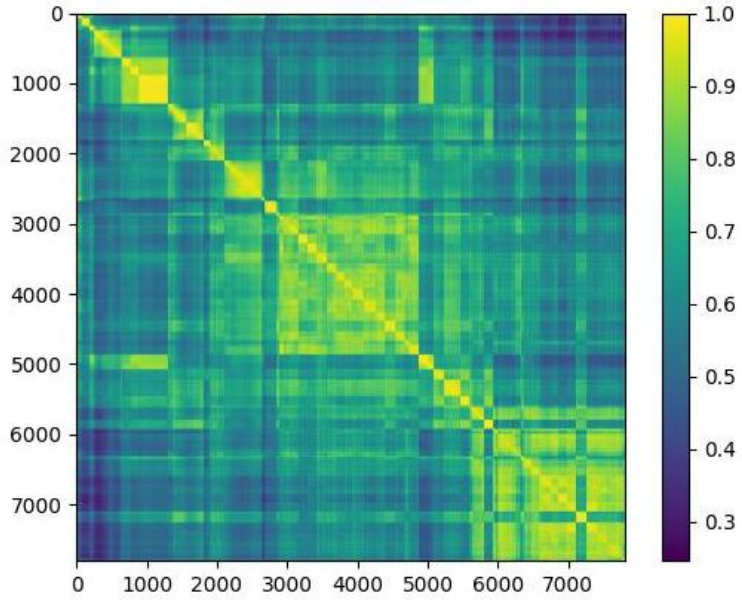

Figura 23: Matriz de distancias coseno.

A continuación, se procederá con el cálculo del gradiente. Al ser una matriz simétrica, sólo es necesario el cálculo en una de las direcciones. Se ha calculado el gradiente a lo largo del eje horizontal, es decir, entre columnas, mediante la función *gradient* de la librería Numpy. Y se ha guardado el valor obtenido en la diagonal de la matriz, que es el que indicará dónde limitan las tomas.

Para la decisión de segmentación, se ha optado por el método de umbralización. Se van a utilizar valores negativos ya que se ha comprobado que los cambios se producen con anterioridad a valores de gradiente negativos.

Se han realizado comprobaciones para diferentes umbrales. Como ejemplo se va a comparar los resultados usando dos umbrales. Uno para un umbral de valor la mitad al valor mínimo del gradiente (representado de color verde en la Figura 24: -0.098), con la intención de realizar una segmentación amplia. Y un umbral de valor un tercio del mínimo (representado de color rojo en la Figura 24: -0.065), para una segmentación más orientada a cambios más ligeros de cámara. A medida que se disminuye el umbral, el número de grupos de *frames* disminuye también.

La variable *threshold*, indicará el valor por el cual se dividirá el valor mínimo del gradiente para calcular el umbral de segmentación. Los fragmentos contendrán mayor número de fotogramas. En la Figura 24, se puede observar el cálculo del gradiente de la matriz representada en la Figura 23, junto a los dos umbrales de decisión que se han seleccionado para las diferentes búsquedas.

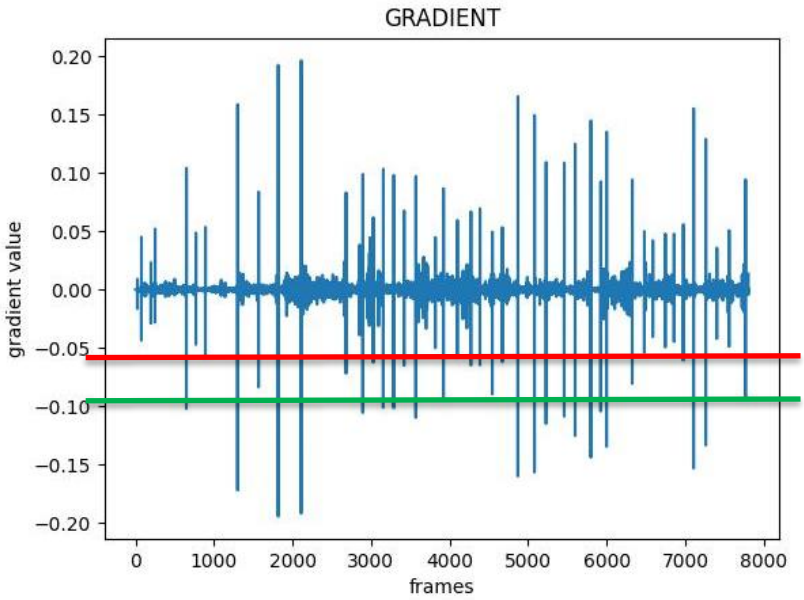

Figura 24: Método del cálculo del gradiente y umbralización.

Una vez segmentado el vídeo, es decir, guardado los valores de inicio y fin de las escenas, se ha implementado el proceso de indexación (Anexo I.4). Se han extraído, además, el *path* del vídeo y para el *embedding vector*, se ha escogido el fotograma central del conjunto para tomarlo como descripción de la escena. Después, se han introducido los datos en la base de datos Milvus.

Tras el almacenamiento de los datos, se ha continuado con el proceso de búsqueda. Para ello, se ha utilizado el mismo modelo CLIP, el 'RN50', para transformar texto de entrada a un *embedding vector*. Esto permitirá comparar en el mismo espacio dos tipos de datos que anteriormente era imposible comparar.

Teniendo en cuenta que los vectores con los que estamos trabajando han sido normalizados desde un principio, el vector consulta también debe ser normalizado. A partir de ahí, la base de datos Milvus, admite realizar búsquedas analizando el producto interno entre vectores, mediante la Ecuación 3. "Si usas IP para calcular similitudes de *embedding vectors*, debes normalizar tus *embeddings*. Después de la normalización, el producto interno es igual a la similitud del coseno." (Similarity Metrics., 2022) [40].

$$p(A, B) = A \cdot B = \sum_{i=1}^{n} a_i \times b_i \tag{3}$$

4.2. Resultados

Realizando una serie de búsquedas (Anexo I.6) se ha corroborado que el sistema funciona. Se ha implementado una función en Python que muestra por pantalla el segmento donde se ha encontrado la consulta, acompañado del fotograma inicial y final del segmento junto al resultado del porcentaje de la similitud obtenido en Milvus.

Por ejemplo, se ha utilizado el video del anterior apartado: *'video_bbc_01.mp4'*, e introduciendo como parámetros: el modelo CLIP para la transformación del dominio visual al dominio vectorial, la colección donde se va a almacenar, *'frames_collection_bbc_01_V5'*, la frecuencia de muestreo la del propio video y el *threshold* = 2. En la Figura 25, pueden observarse los segmentos por los que el algoritmo ha decidido fragmentar.

```
[{'video_id': 1, 'video_start': 0, 'video_stop': 648},
 {'video_id': 2, 'video_start': 649, 'video_stop': 1300},
 {'video_id': 3, 'video_start': 1301, 'video_stop': 1817},
 {'video_id': 4, 'video_start': 1818, 'video_stop': 2109},
 {'video_id': 5, 'video_start': 2110, 'video_stop': 2894},
 {'video_id': 6, 'video_start': 2895, 'video_stop': 3155},
 {'video_id': 7, 'video_start': 3156, 'video_stop': 3288},
 {'video_id': 8, 'video_start': 3289, 'video_stop': 3569},
 {'video_id': 9, 'video_start': 3570, 'video_stop': 4869},
 {'video_id': 10, 'video_start': 4870, 'video_stop': 5080},
 {'video_id': 11, 'video_start': 5081, 'video_stop': 5228},
 {'video_id': 12, 'video_start': 5229, 'video_stop': 5459},
 {'video_id': 13, 'video_start': 5460, 'video_stop': 5598},
 {'video_id': 14, 'video_start': 5599, 'video_stop': 5798},
 {'video_id': 15, 'video_start': 5799, 'video_stop': 5923},
 {'video_id': 16, 'video_start': 5924, 'video_stop': 5998},
 {'video_id': 17, 'video_start': 5999, 'video_stop': 7106},
 {'video_id': 18, 'video_start': 7107, 'video_stop': 7261},
 {'video_id': 19, 'video_start': 7262, 'video_stop': 7806}]
```

Figura 25: Segmentos localizados.

Para la comprobación de los segmentos, se ha observado si los segmentos presentan un contenido parecido dentro del rango de fotogramas y si existe un cambio entre dos secuencias contiguas (Anexo I.5). En la Figura 26, puede verse un claro ejemplo de la existencia de un cambio de plano.

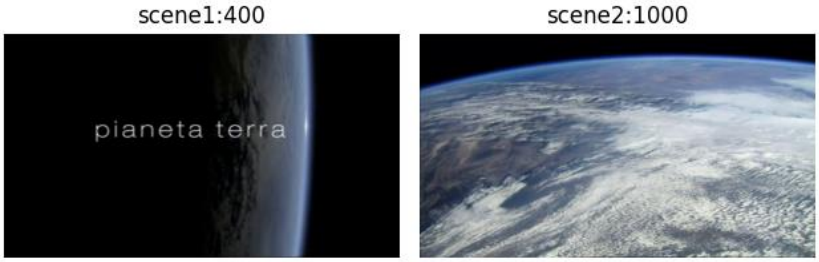

Figura 26: Cambio de plano entre escenas contiguas.

Como ejemplo, para la Figura 27 y para la Figura 28 se ha realizado la búsqueda del texto: *'Polar bears in the snow.'*. Milvus busca hasta *nr* escenas que más se parecen a dicho texto y recupera la información. Siendo *nr* = 4 para esta prueba, puede observarse, que la consulta corresponde con los resultados obtenidos. La variable *score*, devuelve la estimación de similitud que ha realizado Milvus.

Figura 27: Búsqueda de 'Polar bears in the snow.' Con threshold = 2.

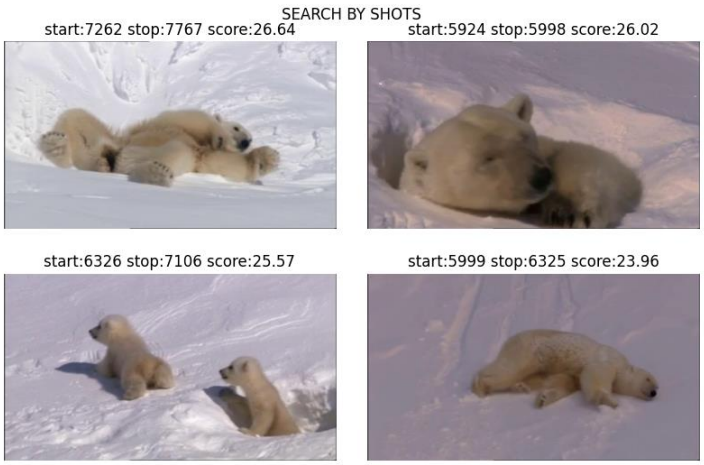

Figura 28: Búsqueda de 'Polar bears in the snow.' Con threshold = 3.

En el caso de que no exista ningún resultado en la búsqueda, aparecerá por pantalla el siguiente mensaje: 'No matches found'.

En el caso de que se escoja un umbral muy negativo, se ha podido comprobar que llega un punto que se podría segmentar el vídeo por secuencias de contextos totalmente diferentes. Como es el caso de utilizar el umbral de valor -0.13 en este ejemplo. En la Figura 29, puede verse la segmentación realizada y en la Figura 30, la definición de dos secuencias consecutivas.

```
 [{'video_id': 1, 'video_start': 0, 'video_stop': 1300},
{'video_id': 2, 'video_start': 1301, 'video_stop': 1817},
{'video_id': 3, 'video_start': 1818, 'video_stop': 2109},
{'video_id': 4, 'video_start': 2110, 'video_stop': 4869},
{'video_id': 5, 'video_start': 4870, 'video_stop': 5080},
{'video_id': 6, 'video_start': 5081, 'video_stop': 5798},
{'video_id': 7, 'video_start': 5799, 'video_stop': 5998},
{'video_id': 8, 'video_start': 5999, 'video_stop': 7106},
{'video_id': 9, 'video_start': 7107, 'video_stop': 7261},
{'video_id': 10, 'video_start': 7262, 'video_stop': 7806}]
```

Figura 29: Segmentación con threshold = 1.5.

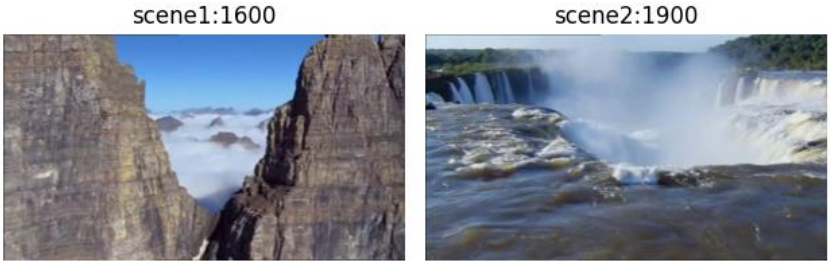

Figura 30: Secuencias consecutivas con threshold =1.5.

Tras repetir el mismo procedimiento para una secuencia de vídeos, se ha planteado el desarrollo de una aplicación que aproveche las búsquedas realizadas para crear una composición (Anexo I.7).

La aplicación en cuestión deberá concatenar los fotogramas que componen las escenas de video buscadas. Para ello, se ha implementado una función que permite mediante la entrada de varios rangos de fotogramas, extraer de cada *path* las imágenes que componen a dicha escena en cuestión.

Como entrada al algoritmo se debe indicar el inicio y final de los segmentos de video que se han dado como solución en el paso anterior. El algoritmo selecciona en orden secuencial las escenas, las concatena a una frecuencia de muestreo de 25 fotogramas por segundo. Finalmente, la concatenación de fotogramas da como resultado un nuevo video.

Como ejemplo, se ha creado una nueva composición juntando dos escenas de vídeos diferentes: *'video_bbc_01.mp4'* y *'video_bbc_02.mp4'*. Para ambos vídeos se

ha seleccionado la colección de escenas con *threshold* = 2. Del primer video se ha escogido la escena con mayor similitud al texto evaluado anteriormente: *'Polar bears in the snow.'* Figura 31. Y para el segundo video se ha tomado el mejor resultado de la búsqueda: *'Lava volcano.'*, Figura 32.

Figura 31: Escena 1 'Polar bears in the snow.'

Figura 32: Escena 2 'Lava volcano.'

Como resultado se ha obtenido un video en el cual aparece como primera escena, la escena 1 y como segunda escena, la escena 2. En la Figura 33, se han realizado dos capturas de la composición de la nueva creación: *'new_creation.mp4'*.

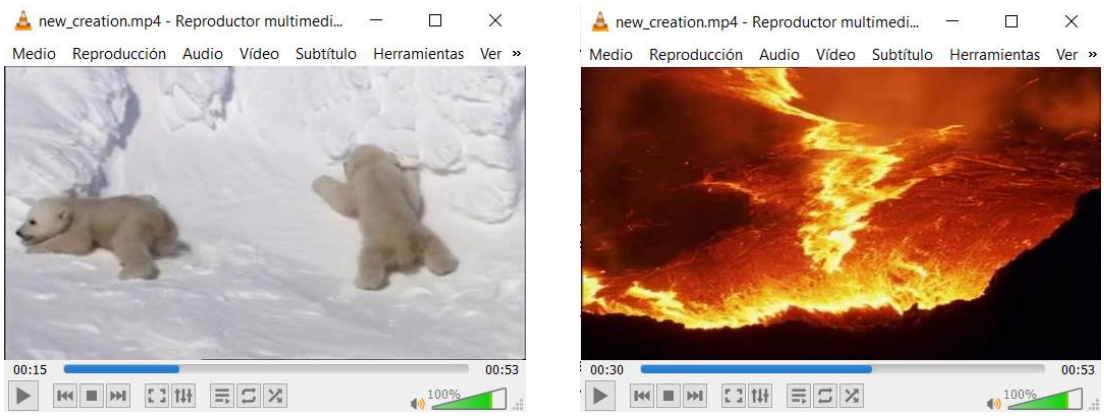

Figura 33: Nueva creación.

Capítulo 5

Conclusiones y Líneas futuras

5.1. Conclusiones

La finalidad de este trabajo ha consistido en estudiar las diferentes posibilidades que ofrece la búsqueda y la recuperación de la información multimedia.

En primer lugar, se ha estudiado la procedencia de esta necesidad en el mundo actual. Además de las herramientas y tecnologías que se han ido desarrollando a lo largo de los años, para buscar una solución óptima e innovadora.

Tras haber identificado el tipo de dato a tratar, y en base a sus características y funciones que se querían implementar, se han escogido las herramientas que conforman la aplicación. Y con ello, se han seleccionado los métodos que más se adecuaban a los requisitos del proyecto.

Se ha utilizado el modelo 'RN50' de CLIP para la transformación de imágenes y texto a *embedding vectors*. La librería OpenCV, para la extracción de metadatos. Un método de segmentación semántica basado en el cálculo de la distancia coseno entre vectores, combinado con el cálculo del gradiente para que mediante umbralización, se puedan detectar los límites de escenas dentro de los vídeos. La base de datos Milvus para el almacenamiento de las escenas y la realización de búsquedas mediante la medida de similitud basada en el cálculo del producto interno. El IP se realiza entre el vector consulta, vector que representa el texto de entrada, y los vectores almacenados en la base de datos, que corresponden con la imagen que identifica al grupo de fotogramas de cada escena.

Una vez, pasado los procesos al lenguaje de programación Python, se han realizado una serie de pruebas utilizando diferentes umbrales de decisión y diferentes textos de consulta, para verificar el funcionamiento del sistema. El sistema engloba cada uno de los ficheros Python, cuyos nombres hacen referencia a los procesos.

El estudio de los resultados se ha enfocado en la comprobación de la segmentación de escenas. Para ello, se ha mostrado por pantalla los rangos de fotogramas que componen cada escena y se ha visualizado que se produce un cambio entre dichos fotogramas consecutivos.

Por último, se ha comprobado que el sistema pueda realizar búsquedas correctamente, introduciéndole diferentes textos de entrada.

En base al trabajo y las pruebas realizadas, se ha concluido que el sistema permite realizar todas las tareas por las que ha sido diseñado. El sistema segmenta vídeos de diferentes características mediante un procedimiento de segmentación semántica, indexa las escenas y permite al usuario realizar consultas y recuperar la información que estaba buscando.

5.2. Líneas futuras

Las redes neuronales y las bases de datos no relacionales son tecnologías relativamente nuevas, por lo que podrían añadirse mejoras al sistema para explotar las funcionalidades que ofrecen.

Podrían explorarse los diferentes métodos de segmentación, o realizar combinaciones, para extraer otro tipo de conclusiones. Así como comparar con otro tipo de medidas de similitud.

En cuanto a la base de datos Milvus podría implementarse la versión Milvus cluster, para comparar las ventajas o desventajas de usar *clusters* mediante Kubernetes. (K8s). [22]

Otro punto de mejora sería realizar una implementación automática de la aplicación de creación de vídeos. En la que la entrada fuera una descripción textual, y el usuario pudiera elegir el orden de las secuencias. En este proyecto se ha realizado un ejemplo de combinación de dos escenas. La idea que se plantea en el futuro es la de modificar el algoritmo para conseguir combinaciones de numerosas escenas.

Bibliografía

[1] *¿Qué es una red neuronal?* (2023). Obtenido de TIBCO: https://www.tibco.com/es/reference-center/what-is-a-neural-network

[2] *About.* (2023). Obtenido de OpenCV: https://opencv.org/about/

[3] *Algoritmo de similitud de coseno.* (Junio de 2015). Obtenido de Grapheverywhere: https://www.grapheverywhere.com/algoritmo-de-similitud-de-coseno/

[4] Arregui, M. d. (26 de Septiembre de 2022). *¿Qué diferencia una base de datos relacional vs una no relacional?* Obtenido de OBS Business School: https://www.obsbusiness.school/blog/que-diferencia-una-base-de-datos-relacional-vs-una-no-relacional#

[5] Azzopardi, L. (2006). Incorporating context within the language modeling approach for ad hoc information retrieval. *ACM SIGIR Forum., 40*(1), 70-70. doi:https://doi.org/10.1145/1147197.1147211

[6] *Build an Index.* (20 de Diciembre de 2022). Obtenido de Milvus: https://milvus.io/docs/build_index.md

[7] Bush, V. (Julio de 1945). As We May Think. *Atlantic Monthly, 176*, 101-108.

[8] Caldera-Serrano, J., & Caro-Castro, C. (2018). Segmentación de vídeos informativos en televisión: de la práctica profesional a la identificación automática. *Cuadernos de Documentación Multimedia, 30*, 1-17. doi:https://doi.org/10.5209/CDMU.62805

[9] Cárdenas, M., Delgado Fernández, M., & Ramírez Céspedes, Z. (Enero de 2011). Search and Retrieval of Information in the Internet from the Users's Perspective, in Higher Education. *Pedagogía Universitaria, 16*, 70-87.

[10] Castillo, E., Gutiérrez, J., & Hadi, A. (1997). Rule-Based Expert Systems. En *Expert Systems and Probabilistic Network Models.* New York: Springer New York. doi:https://doi.org/10.1007/978-1-4612-2270-5_2

[11] *Cisco Visual Networking Index Predicts Annual Internet Traffic to Grow More Than 20 Percent (reaching 1.6 Zettabytes) by 2018*. (10 de Junio de 2014). Obtenido de Cisco The Newsroom: https://newsroom.cisco.com/c/r/newsroom/en/us/a/y2014/m06/cisco-visual-networking-index-predicts-annual-internet-traffic-to-grow-more-than-20-percent-reaching-1-6-zettabytes-by-2018.html

[12] *CLIP: Connecting Text and Images.* (5 de Enero de 2021). (OpenAI) Obtenido de https://openai.com/blog/clip/

[13] *Create a Collection.* (15 de Diciembre de 2022). Obtenido de Milvus: https://milvus.io/docs/create_collection.md

[14] *Docker Desktop – the fastest way to containerize applications.* (2023). Obtenido de Docker: https://www.docker.com/products/docker-desktop/

[15] *Docker Desktop WSL 2 backend on Windows.* (2013-2023). Obtenido de Docker: https://docs.docker.com/desktop/windows/wsl/

[16] Fernández, J., Miranda, N., Guerrero, R., & Piccoli, F. (2010). Datos no Estructurados No Textuales: Desarrollo de Nuevas Tecnologías. 330-336.

[17] Gil, P., Torres, F., & Ortiz Zamora, F. (2004). Detección de objetos por segmentación multinivel combinada de espacios de color.

[18] Hu, W., Xie, N., Li, L., Zeng, X., & Maybank, S. (6 de Noviembre de 2011). A survey on visual content-based video indexing and retrieval. *IEEE Transactions Systems Man and Cybernetics − Part C: Applications and Reviews., 41*(6), 797-819.

[19] Husain, S. F. (Julio de 2016). Depth Image and Video Segmentation. *Perceiving Dynamic Environments: From Surface Geometry to Semantic Representation.*, págs. 15-38.

[20] *Indexes supported in Milvus.* (15 de Diciembre de 2022). Obtenido de Milvus: https://milvus.io/docs/index.md

[21] *Instalación de Linux en Windows con WSL.* (18 de Enero de 2023). Obtenido de Microsoft: https://learn.microsoft.com/es-es/windows/wsl/install

[22] *Install Milvus Cluster with Milvus Operator.* (16 de Noviembre de 2022). Obtenido de Milvus: https://milvus.io/docs/v2.1.x/install_cluster-milvusoperator.md

[23] Jiménez González, A. (2009). Algoritmos de segmentación por color. *Técnicas de percepción activa para seguimiento de objetos mediante robots móviles en entornos urbanos.*

[24] Kim, J. W. (Enero de 2021). *Model Card: CLIP.* Obtenido de GitHub: https://github.com/openai/CLIP/blob/main/model-card.md

[25] Lascano, X., Pombo, V., & Chavez-Burbano, P. (2011). Implementación de Interfaz Gráfica para Comparación Visual de Métodos de Segmentación y Procesamiento de Video, Usando Matlab.

[26] Lefèvre, S., Holler, J., & Vincent, N. (2003). A review of real-time segmentation of uncompressed video sequences for content-based search and retrieval. *Real-Time Imaging, 9*(1), 73-98. doi:https://doi.org/10.1016/S1077-2014(02)00115-8

[27] Macarrón, P. (25 de Enero de 2021). *Azure y bases de datos no relacionales.* Obtenido de Certia: https://www.certia.net/azure-y-bases-de-datos-no-relacionales/

[28] Macarrón, P. (8 de Marzo de 2021). *Tipos de bases de datos no relacionales.* Obtenido de Certia: https://www.certia.net/tipos-de-bases-de-datos-no-relacionales/

[29] Macarrón, P. (17 de Febrero de 2021). *Tipos de Datos no relacionales.* Obtenido de Certia: https://www.certia.net/tipos-de-datos-no-relacionales/

[30] *Main Components.* (2 de Septiembre de 2022). Obtenido de Milvus: https://milvus.io/docs/main_components.md

[31] *Manage Milvus Connections.* (17 de Agosto de 2022). Obtenido de Milvus: https://milvus.io/docs/manage_connection.md

[32] Martín, M. (21 de Mayo de 2002). Técnicas Clásicas de Segmentación de Imagen.

[33] Mendoza Olguín, G., Laureano De Jesús, Y., & Pérez de Celis Herrero, M. (2019). Métricas de similaridad y evaluación para sistemas de recomendación de filtrado colaborativo. *Revista de Investigación en Tecnologías de la Información., 7*(14), 224-240. doi:https://doi.org/10.36825/RITI.07.14.019

[34] Moreno, A. (17 de Octubre de 2022). *Procesamiento del lenguaje natural ¿qué es?* Obtenido de Instituto de Ingeniería del Conocimiento: https://www.iic.uam.es/inteligencia/que-es-procesamiento-del-lenguaje-natural/

[35] Robinson, J. (10 de Enero de 2022). *Video Search Engine Using OpenAI's CLIP.* Obtenido de Stormin' The Castle: https://www.storminthecastle.com/posts/video_search/

[36] Rodrígez, V. (17 de Octubre de 2018). *Decision trees/Árboles de decisión para clasificar en Python.* Obtenido de https://vincentblog.xyz/posts/decision-trees-arboles-de-decision-para-clasificar-en-python

[37] Salvetti, M. (Septiembre de 2009). Selección dinámica de pivotes que se adaptan a las búsquedas en Espacios Métricos.

[38] Sánchez, J., & Binefa, X. (2000). Color Normalization for Digital Video Processing. En R. Laurini (Ed.), *Advances in Visual Information Systems.* (Vol. 1929, págs.

189-199). Springer, Berlin, Heidelberg. doi:https://doi.org/10.1007/3-540-40053-2_17

[39] Sarkar, D. (27 de Junio de 2022). *Scalable and Blazing Fast Similarity Search With Milvus Vector Database.* Obtenido de Towards AI: https://pub.towardsai.net/scalable-and-blazing-fast-similarity-search-with-milvus-vector-database-d221706e605a

[40] *Similarity Metrics.* (2 de Septiembre de 2022). Obtenido de Milvus: https://milvus.io/docs/v2.1.x/metric.md

[41] Sin, J. R. (Febrero de 2011). Reconocimiento automático de áreas de interés en secuencias de interiores.

[42] Singhal, A. (2001). Modern Information Retrieval: A Brief Overview. *IEEE Data Eng. Bull., 24*, 35-43.

[43] Tech, T. I. (2023). *Datos no estructurados.* Obtenido de Telefónica Tech. AI of Things: https://aiofthings.telefonicatech.com/recursos/datapedia/datos-no-estructurados

[44] *Tipos de Machine Learning.* (Junio de 2015). Obtenido de Grapheverywhere: https://www.grapheverywhere.com/tipos-machine-learning/

[45] *Vector database built for scalable similarity search.* (2023). (LF AI & Data Foundation) Obtenido de Milvus: https://milvus.io/

[46] *Ventajas y desventajas en una base de datos relacional.* (28 de Enero de 2022). Obtenido de Codespace: https://codespaceacademy.com/blog/ventajas-y-desventajas-base-de-datos-relacional/

[47] *What is Milvus vector database?* (31 de Agosto de 2022). Obtenido de Milvus: https://milvus.io/docs/overview.md

Anexo I

I.1. Descarga del modelo CLIP

- Código Python: **CLIPmodel.py**

```
import logging
import clip
import torch

class ClipModel:
  def __init__(self):
    self.device = 'cuda' if torch.cuda.is_available() else 'cpu'
    logging.info("Loading CLIP Model")
    self.clip_model,self.clip_prep =
clip.load('RN50',self.device,jit=False)

model = ClipModel()
```

I.2. Funciones para la extracción de metadatos

- Código Python: **video.py**

```
import cv2
import matplotlib.pyplot as plt

# generator yielding a video frame and a frame
# timestamp (seconds) each time
def video_frames(path):
  video = cv2.VideoCapture(path, apiPreference=cv2.CAP_FFMPEG)
  fps = video.get(cv2.CAP_PROP_FPS)
  ret, frame = video.read()
  count = 0
  while ret:
    count = count + 1
    frame = cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
    yield frame,count/fps
    ret, frame = video.read()

# returns duration of the specified video
def video_duration(path):
  video = cv2.VideoCapture(path, apiPreference=cv2.CAP_FFMPEG)
  fps = video.get(cv2.CAP_PROP_FPS)
```

```
  frame_count = video.get(cv2.CAP_PROP_FRAME_COUNT)
  return frame_count/fps

# returns frame speed
def video_fps(path):
  video = cv2.VideoCapture(path, apiPreference=cv2.CAP_FFMPEG)
  fps = video.get(cv2.CAP_PROP_FPS)
  return fps

# returns a single video frame at the specified timestamp
(seconds)
def video_frame(path,timestamp=0):
  video = cv2.VideoCapture(path, apiPreference=cv2.CAP_FFMPEG)
  video.set(cv2.CAP_PROP_POS_MSEC, timestamp * 1000)
  ret,frame = video.read()
  frame = cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
  return frame if ret else None
```

I.3. Método de segmentación

- Código Python: **segmentation.py**

```
import math
import matplotlib.pyplot as plt
import numpy as np
import video
from scipy import spatial

def function_distance(path, array, time, collection_name,
threshold):
    n_frames = len(time)
    similarity = np.zeros((n_frames, n_frames)) # n_frames x
n_frames matrix with the cosine distance between frame i and
frame j
    # We calculate the main diagonal and the upper triangular
matrix, since it is symmetric, we add the corresponding results
to the lower diagonal.
    for i in range(0,n_frames):
        rows = np.zeros(n_frames)
        for j in range(i,n_frames):
            rows[j] = (1-
spatial.distance.cosine(array[i],array[j])) # cosine distance
        similarity[i] = rows
    similarity = similarity + np.triu(similarity, 1).T
    gradiente = np.diagonal(np.gradient(similarity, axis=0)) #
gradient along axis 0
    plt.figure()
```

```
    plt.plot(gradiente)
    plt.xlabel('frames')
    plt.ylabel('gradient value')
    plt.title('GRADIENT')
    plt.savefig("gradient_{}.jpg".format(collection_name))
    plt.figure()
    plt.imshow(similarity)
    plt.colorbar()
    plt.savefig("similarity_{}.jpg".format(collection_name))

    min = np.min(gradiente)
    distancia = []
    start = 0
    save_start = []
    save_stop = []
    save_embedding = []
    save_path = []
    num = 0
    for i in range(len(time)):
      stop = i-1
      if gradiente[i]<min/threshold and num!=0: # threshold
          distancia.append([str(path),"scene_" + str(start),
start, stop]) # stores start and stop time
          save_start.append(distancia[-1][2])
          save_stop.append(distancia[-1][3])
          save_embedding.append(array[int((start+stop)/2),:])
          save_path.append(distancia[-1][0])
          start = i
          num = 0
      num =+ 1
    distancia.append([path,"scene_" + str(start), start, i+1])
    save_start.append(distancia[-1][2])
    save_stop.append(distancia[-1][3])
    save_embedding.append(array[int((start+stop)/2),:])
    save_path.append(distancia[-1][0])

    return save_start, save_stop, save_embedding, save_path
```

I.4. Indexación

- Código Python: **index.py**

```
import logging
import math
import numpy as np
import progressbar
import torch
```

```
from PIL import Image
from pymilvus import Collection, connections
from segmentation import function_distance
from CLIPmodel import model
from video import video_duration, video_fps, video_frames

def store_frames(model,path,collection_name,freq,threshold):

  last_index = -freq # to take the first frame
  duration = video_duration(path)
  logging.info('Indexing: {}'.format(path))
  # index patches
  time = []
  list_features = None
  list_path = []
  with progressbar.ProgressBar(max_value=math.ceil(duration)) as
progress_bar:
    for frame,timestamp in video_frames(path):
      if timestamp - last_index >= freq-0.001: # sample
        progress_bar.update(math.floor(timestamp))
        last_index = timestamp
        frame_features = []
        frame_path = []
        # clip wants PIL image objects
        pils = []
        frame_path.append(path)
        time.append(np.float32(timestamp))
        pils.append(model.clip_prep(Image.fromarray(frame)))
        # put the image into a single tensor
        tensor = torch.stack(pils,dim=0)
        uploaded = tensor.to(model.device)
        # ask CLIP to encode the image features for our image
into a feature vector
        with torch.no_grad():
          frame_features =
model.clip_model.encode_image(uploaded)
        assert(frame_features.shape[0] == len(frame_path))
        # normalize the image feature vectors so that they all
have a length of 1
        frame_features /= frame_features.norm(dim=-
1,keepdim=True)
        if list_features is not None:
          list_features =
torch.cat((list_features,frame_features),dim=0)
        else:
          list_features = frame_features
        list_path.extend(frame_path)
  array = list_features.numpy()
```

```
  # segmentation process
  save_start, save_stop, save_embedding, save_path =
function_distance(path, array, time, collection_name, threshold)
  # connection to Milvus server
  connections.connect("default", host="localhost", port="19530")
  collection = Collection(collection_name)
  data = [list(range(1,len(save_embedding)+1)), save_embedding,
save_start, save_stop, save_path]
  # insert data into collection
  collection.insert(data)
  # build an index
  index_params = {"metric_type": "IP", "index_type":
"FLAT","params": {}}
  collection.create_index(field_name="video_vector",
index_params=index_params)

path = 'video_bbc_01.mp4'
collection_name = "frames_collection_{}".format("bbc_01_V5")
threshold = 2
store_frames(model,path,collection_name,1/video_fps(path),thresh
old)
```

I.5. Comprobación de la segmentación

- Código Python: **segmentation_verification.py**

```
import numpy as np
from matplotlib import pyplot as plt
from pymilvus import Collection, connections
from video import video_duration, video_fps, video_frame

def show_results(scene1, scene2, video_path):
  fig,ax = plt.subplots(1,2)
  a = ax.ravel()
  time1=np.float32(scene1/video_fps(video_path))
  frame1 = video_frame(path=video_path,timestamp=time1)
  a[0].imshow(frame1)
  a[0].set_title('scene1:{}'.format(scene1))
  time2=np.float32(scene2/video_fps(video_path))
  frame2 = video_frame(path=video_path,timestamp=time2)
  a[1].imshow(frame2)
  a[1].set_title('scene2:{}'.format(scene2))
  for ax in fig.axes:
    ax.axis("off")
    plt.tight_layout()
  plt.show()
```

```
path = 'video_bbc_01.mp4'
connections.connect("default", host="localhost", port="19530")
collection_name = "frames_collection_{}".format("bbc_01_V5")
collection = Collection(collection_name) # Get an existing
collection.
collection.load()
shape =
list(range(1,1+int(video_fps(path)*video_duration(path))))

result = collection.query(
  expr = "video_id in {}".format(shape),
  offset = 0,
  limit = 65535,
  output_fields = ["video_id","video_start","video_stop"],
  consistency_level="Strong"
)

result = sorted(result, key=lambda k: k["video_id"])

print(result)

scene1 = 400
scene2 = 1000
show_results(scene1, scene2, path)
```

I.6. Búsqueda

- Código Python: **search.py**

```
import math
import clip
import numpy as np
import torch
from matplotlib import pyplot as plt
from pymilvus import Collection, connections
from CLIPmodel import model
from video import video_fps, video_frame

def search(model,query,collection,nr):
  # ask CLIP to encode our query into a feature vector
  query_tensor =
torch.cat([clip.tokenize(query)]).to(model.device)
  with torch.no_grad():
    query_features = model.clip_model.encode_text(query_tensor)
  # normalize the query feature vector so that it has a length
of 1
  query_features /= query_features.norm(dim=-1,keepdim=True)
```

```
  # do the actual search here by calculating the distances
between the query vector
  # and all of the image features from our video with a single
dot product
  # lets pull out the best matches
  collection.load()
  search_params = {"metric_type": "IP"}
  array = query_features.numpy()
  result = collection.search(array, anns_field="video_vector",
param=search_params, limit=nr, expr=None, output_fields =
["video_id", "video_path", "video_start", "video_stop"])
  print(result)
  searched_score = 100.0*np.array(list(result[0].distances))
  print(searched_score)
  result_frames = [[None] * np.shape(result)[1] for i in
range(5)]
  for hits in result:
    for j,hit in enumerate(hits):
        result_frames[0][j] = hit.entity.get('video_id')
        result_frames[1][j] = searched_score[j]
        result_frames[2][j] = hit.entity.get('video_path')
        result_frames[3][j] = hit.entity.get('video_start')
        result_frames[4][j] = hit.entity.get('video_stop')
  return result_frames

def show_results(matches):
  if len(matches) > 0:
    rows = int(math.ceil(len(matches[0])/2))
    fig,ax = plt.subplots(rows,2,figsize=(10,rows*2))
    a = ax.ravel()
    for i in range(len(matches[0])):
      time=np.float32(int((matches[3][i]+matches[4][i])/2)/video
_fps(matches[2][i]))
      print(int((matches[3][i]+matches[4][i])/2))
      frame = video_frame(path=matches[2][i],timestamp=time)
      a[i].imshow(frame)
      a[i].set_title('start:{} stop:{}
score:{}'.format(matches[3][i],matches[4][i],round(matches[1][i]
,2)))
    for ax in fig.axes:
        ax.axis("off")
    plt.tight_layout()
  else:
    print('No matches found')

connections.connect("default", host="localhost", port="19530")
collection_name = "frames_collection_{}".format("bbc_01_V5")
collection = Collection(collection_name)
```

```
nr = 4 # limit number of results
result_frames = search(model,'Polar bears in the
snow',collection,nr)

connections.connect("default", host="localhost", port="19530")
collection = Collection(collection_name) # Get an existing
collection.
collection.load()
shape = list(range(1,5))

result = collection.query(
  expr = "video_id in {}".format(shape),
  offset = 0,
  limit = 65535,
  output_fields =
["video_id","video_start","video_stop","video_path"],
  consistency_level="Strong"
)

result = sorted(result, key=lambda k: k["video_id"])

print(result_frames)
show_results(result_frames)
plt.suptitle('SEARCH BY SHOTS')
plt.show()
```

I.7. Creación de vídeos en base a escenas

- Código Python: **video_creation.py**

```
import numpy as np
import cv2

# segment about polar bears
time1 = [i for i in range(7262,7806+1)]
# segment about lava volcano
time2 = [i for i in range(3067, 3854+1)]

def video_fps(path):
  video = cv2.VideoCapture(path, apiPreference=cv2.CAP_FFMPEG)
  fps = video.get(cv2.CAP_PROP_FPS)
  return fps

def video_frame(path,timestamp=0):
  video = cv2.VideoCapture(path, apiPreference=cv2.CAP_FFMPEG)
  video.set(cv2.CAP_PROP_POS_MSEC, timestamp * 1000)
  ret,frame = video.read()
```

```
  return frame if ret else None

def video_scene(path,timestamp):
    frame = []
    for i in range(len(timestamp)):
        frame.append(video_frame(path,timestamp[i]))
    return frame

def video_writer(path1, path2, new_path,timestamp1, timestamp2):
  frame = video_scene(path1,timestamp1)
  frame.extend(video_scene(path2,timestamp2))
  fps = 25
  height, width  = frame[0].shape[:2]
  video = cv2.VideoWriter(new_path,cv2.VideoWriter_fourcc('m',
'p', '4', 'v'),fps,(width,height))
  for i in range(len(frame)):
    video.write(frame[i])
  video.release() # release resources

path1 = "video_bbc_01.mp4"
path2 = "video_bbc_02.mp4"
for i in range(len(time1)):
    time1[i]=np.float32(int(time1[i])/video_fps(path1))
for i in range(len(time2)):
    time2[i]=np.float32(int(time2[i])/video_fps(path2))
video_writer(path1, path2, new_path='new_creation.mp4',
timestamp1 = time1, timestamp2 = time2)
```