UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTACIÓN Y COMPARACIÓN DE DESCRIPTORES PARA BÚSQUEDA EN VIDEO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

DIEGO ANDRÉS DÍAZ-ESPINOZA

PROFESOR GUÍA:
BENJAMÍN BUSTOS CÁRDENAS

MIEMBROS DE LA COMISIÓN:
GONZALO NAVARRO BADINO
RODRIGO PAREDES MORALEDA

SANTIAGO DE CHILE
ABRIL 2010

ESTE TRABAJO HA SIDO FINANCIADO POR EL PROYECTO
FONDECYT 11070037

RESUMEN DE LA MEMORIA PARA OPTAR AL TITULO
DE INGENIERO CIVIL EN COMPUTACION

POR: DIEGO DIAZ-ESPINOZA

FECHA: 16/04/2010

PROFESOR GUIA: Sr. BENJAMIN BUSTOS CARDENAS

Implementación y Comparación de descriptores para Búsqueda en Video

Hoy en día existe una gran cantidad de videos en formato digital, al igual que un gran interés por utilizar estos medios en ambitos tan diversos que van de las ciencias a la entretención. Junto con reproducir videos y para obtener el máximo de beneficios de esta enorme cantidad de información digital, se necesita de métodos efectivos para consultar y navegar en ellos. Esto permite implementar aplicaciones como buscadores en colecciones de videos, detectores de reproducciones ilegales en canales de televisión, gestores de contenidos para edición de video, entre otras. Una manera de abordar este problema de búsqueda, es mediante el uso de *descriptores de video*, los cuales han surgido en investigaciones desde hace algunas décadas atrás. Los descriptores de video son funciones matemáticas que reducen la información contenida en el video, manteniendo el máximo de información relevante y generalmente se expresan mediante vectores sobre algún espacio matemático particular. Esto permite un orden relativo de los datos, facilitando la posterior búsqueda. Actualmente existen muchos descriptores en distintas publicaciones científicas, todos ellos con distintas características y obteniendo distintos grados de efectividad al momento de realizar búsquedas en colecciones. Esta memoria pretende estudiar, implementar y comparar un sub-conjunto de descriptores -aparecidos en publicaciones recientes-, utilizando medidas de correctitud, eficiencia, eficacia y robustez a través de experimentación.

For nearly forty years this story has given faithful service to the Young in Heart; and Time has been powerless to put its kindly philosophy out of fashion. To those of you who have been faithful to it in return...and to the Young in Heart...we dedicate this picture.

Victor Fleming*, *The Wizard Of Oz (1939)

El sueño de ser Ingeniero comienza el año 2002 en las salas de estudio de Bachillerato, impulsado fuertemente por los profesores del programa y maestros de computación –en estudios anteriores– como Juan Carlos Letelier y Jaime Catrileo. El año 2004 tomo mi primer ramo de computación en la facultad al alero de Juan Alvarez, en sus clases decido finalmente mi especialidad. Durante este largo pero lindo sueño me ayudan y apoyan mi padre Carlos (a quién debo el incondicional interés hacia la ciencia e investigación) y mi madre Paulina (a quien debo la importancia de la responsabilidad y dedicación a los estudios). Sin duda los pasos de mi hermano Rodrigo me iluminan y de alguna forma me muestran el futuro y cómo enfrentarse a él en tiempos dificiles. Junto a la compañía de mi querida Luna logro llegar a la meta sin mayores inconvenientes, pero sí con mucho esfuerzo. El sueño entonces queda reflejado en muchas materias aprendidas durante estos seis años, que se juntan por cosas del destino en un tema propuesto por mi profesor guía Benjamín. Para todos ellos muchas gracias.

Índice General

Capítulo 1

Introducción

1.1. Motivación

Cada día hay más videos disponibles para todo tipo de usuarios (ver por ejemplo YouTube[1], GoogleVideo[2], Videos en Yahoo[3]). Esto en parte es fomentado por el decreciente costo de los sistemas de almacenamiento y la creciente producción de medios de captura para video. Al mismo tiempo que la cantidad de videos aumenta, también lo hacen las facilidades para su uso en distintas áreas, como por ejemplo televisión. Además, los videos van penetrando en áreas en que antes no se presentaban, a la vez que se mezclan con anteriores medios de comunicación de datos. Este es el caso de los videos en la Web y dispositivos móviles por ejemplo.

La disponibilidad de más datos, en este caso videos, implica la necesidad de una mejor gestión de los mismos. En este contexto, surge la necesidad de buscar, consultar, indexar y navegar en grandes conjuntos de archivos de video [1]. Para búsqueda en video existen variadas técnicas, las que han sido investigadas desde finales de los 90 [6] [8] [18]. En cuanto a búsqueda, en general se utiliza el concepto de "búsqueda por similitud", que consiste en encontrar objetos, por ejemplo videos, imágenes, audio, etc. que son similares pero no necesariamente idénticos al objeto de consulta.

El enfoque de la búsqueda por similitud presenta el problema de cómo comparar (esto es, medir cuánto se parecen) dos videos. Para resolver esta problemática central de la búsqueda,

[1] http://www.youtube.com/
[2] http://video.google.com/
[3] http://video.yahoo.com/

el proceso ha consistido tradicionalmente en describir la imagen en términos matemáticos. Por ejemplo, se utilizan descriptores [1] (*descriptors*) que son vectores que representan la imagen y bajo los cuales las comparaciones están definidas mediante funciones de distancia. Un ejemplo de descriptor para un video cualquiera consiste en extraer ciertas imágenes del video cada ciertos intervalos de tiempo fijos y calcular lo que se denomina Histograma de Colores [5]. Esto es clasificar cada uno de los pixeles de cada una de las imágenes en una de muchas categorías que representan colores. Cada histograma de las imágenes se representa luego como un vector n-dimensional, donde cada dimensión representa uno de los colores del histograma. Finalmente, el video queda representado por un conjunto de vectores y la búsqueda de similitud entre videos se reduce a calcular las menores distancias entre todos los videos disponibles.

El principal problema presentado por estas técnicas basadas en descriptores es que su variabilidad en la calidad de los resultados obtenidos es alta. Más aún, es posible obtener conjuntos de resultados completamente diferentes usando distintos descriptores pero manteniendo el conjunto de videos sobre el que se busca. Un buen descriptor de video debe retornar videos similares, según el concepto de "similitud humana", el que sin duda es subjetivo.

Esta memoria se centra en la implementación de descriptores para búsqueda en video aparecidos en publicaciones científicas recientes. Adicionalmente se realizará un estudio experimental de los mismos para determinar las ventajas y desventajas de los descriptores implementados. A partir de los resultados obtenidos, se pretende concluir cuáles son los problemas problemas y complejidades de los descriptores implementados, para realizar búsquedas en colecciones de video.

1.2. Objetivos

El objetivo general de esta memoria es estudiar e implementar un conjunto de descriptores para búsqueda en video, y luego realizar una comparación entre ellos mediante *benchmarks*.

Entre los objetivos específicos se tendrá en cuenta:

1. Búsqueda bibliográfica respecto a la búsqueda en video desde el año 1999 a la fecha.

2. Estudio de descriptores para búsqueda en video y sus conceptos básicos asociados.

3. Implementación de 4 descriptores para búsqueda en video. En particular, se implementarán los algoritmos aparecidos en Naturel y Gros [14], Shen *et al.* [17], Iwamoto *et al.* [10] y Kim *et al.* [11].

4. Formalización de un marco experimental que permita comparar descriptores. En este aspecto se considerará:

   a) Implementación de interfaz *ad hoc* para realización de experimentos.

   b) Búsqueda de una colección de videos *ad hoc* para realizar las búsquedas en la fase experimental.

   c) Uso de medidas de efectividad, generación de gráficos y análisis de los mismos.

5. Comparación de la efectividad y eficiencia de los descriptores implementados.

1.3. Estructura de la memoria

El Capítulo 2 propone algunas definiciones útiles, describe los conceptos computacionales de "Imagen" y "Video", define qué se entiende por descriptores y cómo apoyan la búsqueda por similitud, expone los tipos de búsqueda que se pueden utilizar y muestra qué concepto matemático está detrás de las búsquedas y cómo se evalúan los resultados obtenidos. El Capítulo 3 expone los descriptores escogidos para trabajar en esta memoria: "Edge descriptor", "Mean average descriptor", "Bounded Coordinate System descriptor" (BCS) y "Discrete Cosine Transform Based descriptor". El Capítulo 4 detalla el propósito de la implementación, los casos de uso y usuarios, la plataforma de trabajo y termina con una descripción de las clases y su arquitectura. El Capítulo 5 plantea los objetivos de los experimentos para testear la *perfomance* de los descriptores, detalla los experimentos a realizar y muestra los resultados de los mismos junto con un breve análisis. El Capítulo 6 entrega las conclusiones finales y trabajo futuro a realizar.

Capítulo 2

Conceptos básicos

2.1. Definiciones

1. Imagen: Es una representación computacional bi-dimensional de algún objeto o cosa de forma pictórica (en el sentido de representación gráfica usando elementos visuales).

2. Video: Es una secuencia de imágenes (de aquí en adelante *"frames"*). Cada video presenta las imágenes cada cierto intervalo de tiempo (por ejemplo 24 *frames* por cada segundo o fps). Los videos tienen 3-dimensiones: dos espaciales que están dadas por las dimensiones de los *frames* y una temporal, dada por la transición entre *frame* y *frame*. Lo que tarda un cambio de un *frame* a otro se considera como la unidad mínima de tiempo.

3. Clip: Video de no más de 10 minutos de duración [17].

4. Shot: Secuencia de *frames* contiguos, grabados por un dispositivo de captura de video sin cortes. Se extiende también a un conjunto de *frames* con coherencia semántica que actúan como unidad dentro del video [14].

5. Velocidad del video: Es la cantidad de *frames* por segundo que tiene el video. Para usos de cine y televisión, se acostumbra usar 24 fps o 30 fps.

2.2. Representación de imágenes y videos

Dado un espacio de colores $\mathbb{C}$, se define una imagen I como una función:

$$I : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{C} \tag{2.1}$$

La función imagen asocia a cada par ordenado de valores en $\mathbb{Z} \times \mathbb{Z}$ (siendo $\mathbb{Z}$ el conjunto de enteros) un valor en el espacio de colores $\mathbb{C}$. El conjunto $\mathbb{Z}$ está acotado por las dimensiones de la imagen en cuestión. El espacio de colores generalmente es una tripleta de la forma $c = (x, y, z)$. Por ejemplo, para RGB (*Red-Green-Blue*) se tiene que $c = \{r, g, b\}$, donde $r, g, b \in [0, 255]$ o $[0, 1]$ si normalizamos el espacio; en este caso cada componente de la tripleta indica la cantidad de ese color en el total formado por la combinación lineal de los tres. También existen espacios de colores basados en intensidades -como YUV, LUV, YCrCb- en donde una de las componentes acarrea información de luminosidad o intensidad (más cercano al negro o blanco) y las otras dos corresponden a información de color [9].

2.2.1. Procesamiento de imágenes

Al tratar las imágenes como funciones matemáticas, se pueden aplicar transformaciones sobre las mismas. Dado que los videos son un conjunto de imágenes (*frames*) también permiten la aplicación de transformaciones. Una de estas transformaciones son los filtros para imágenes, también llamados filtros de convolución (*convolution*) o filtros de caja (*box filters*).

Filtros de imágenes (Convolución discreta): Se define un *kernel* o "núcleo de convolución" mediante la función G de la Ecuación 2.2, donde $\digamma \subset \mathbb{Z} \times \mathbb{Z}$. El *kernel* es una matriz M cuadrada de tamaño A definida en la Ecuación 2.3, tal que a cada $a(i, j) \quad i, j \in [0, A-1]$ asocia un número en los reales. Cada *kernel* tiene un punto de anclaje que generalmente es el centro de la matriz.

$$G : \digamma \times \digamma \rightarrow \mathbb{R} \tag{2.2}$$

$$M = \left(\begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,A-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,A-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{A-1,0} & a_{A-1,1} & \cdots & a_{A-1,A-1} \end{bmatrix} \right) \tag{2.3}$$

Sea una imagen $I(x, y)$ con $0 \leq x \leq N-1, 0 \leq y \leq M-1$, un *kernel* $G(i, j), \quad i, j \in [0, A-1]$, la convolución $H(x, y)$ se define como $H(x, y)$ (Ecuación 2.4)

$$H(x,y) = \sum_{i=0}^{A} \sum_{j=0}^{A} I\left(x + i - \frac{A}{2}, y + j - \frac{A}{2}\right) G(i,j) \tag{2.4}$$

En términos simples, la función de convolución $H(x, y)$ de la Ecuación 2.4, superpone en cada (x, y) de I el centro del *kernel*, pondera cada uno de los $I(x, y)$ que quedan sobre el *kernel* y los suma obteniendo un valor que es reasignado al valor del pixel (x, y) en la imagen (ver Figura 2.1).

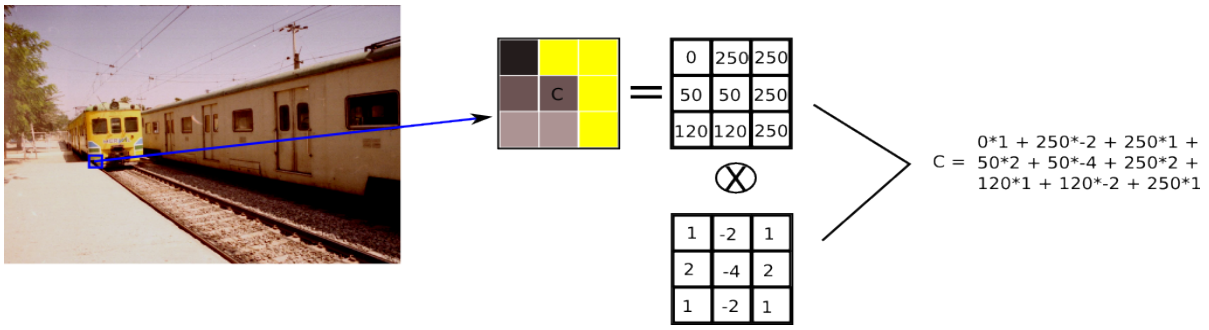

Figura 2.1: Aplicación de un filtro de convolución de *kernel* 3×3 sobre un área particular de una imagen. El punto de anclaje del *kernel* está en el centro de la matriz.

2.2.2. Transformaciones sobre videos

Los videos son tratados como una secuencia ordenada de imágenes (*frames*), de esta forma, cualquier función que se aplique a una imagen, también puede aplicarse a un video.

En particular, sobre cada imagen de un video se pueden aplicar transformaciones tales como ruido (*noise*), reflexión sobre el eje vertical u horizontal (*mirroring*), desenfoque (*blur*), entre otras. Estas transformaciones son muy comunes en medios de transmisión de video (televisón e Internet), en donde algunas se aplican de manera intencional (por ejemplo en la inserción de logotipos corporativos) y otras de forma circunstancial (como el ruido provocado en las imágenes debido a un medio de transmisión con pérdida).

Las transformaciones operan sobre el conjunto de datos que es enviado sin informar al receptor, de modo que el receptor sólo tiene acceso a los datos modificados y no los originales,

haciendo que éstas funciones sean irreversibles. La mayoría de las transformaciones premeditadas se basan en uno o varios parámetros de ajuste, que determinan el nivel de distorsión final sobre el video original.

En la Figura 2.2 se muestran distintos tipos de transformaciones sobre un *frame* de un video. En la Figura 2.3 de la página 11, hay 3 videos (hacia abajo) junto con variaciones en sus parámetros de entrada (hacia la derecha). A continuación se describe un conjunto de transformaciones típicamente utilizadas:

1. ***Picture in Picture:*** **(Pip)** Intenta mostrar lo que ocurre cuando un video se sobrepone dentro de otro en menor escala. El video contiene ahora dos áreas de información semántica: una correspondiente al fondo (*Background*) y otra al primer plano (*Foreground*). El fondo utiliza la mayor parte del dispositivo de salida, mientras que el primer plano utiliza sólo una región acotada de dimensiones menores (por ejemplo $\frac{1}{9}$ de la pantalla).

2. ***Blur:*** Es la aplicación de un *kernel* Gaussiano, que torna la imagen borrosa. Los componentes del *kernel* $G(x, y)$ aplicado, son coeficientes de una función Gaussiana como la mostrada en la Ecuación 2.5. Un ejemplo de *kernel* Gaussiano ($\sigma = \sqrt{2}$) se muestra en la Ecuación 2.6 [15].

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.5}$$

$$K = \left(\begin{bmatrix} 0,05 & 0,07 & 0,05 \\ 0,07 & 0,08 & 0,07 \\ 0,05 & 0,07 & 0,05 \end{bmatrix} \right) \tag{2.6}$$

3. ***Captions:*** Se sobrepone al video alguna imagen pequeña. Intenta imitar lo ocurrido en programas televisivos que insertan logotipos en los flujos de salida. El logotipo tiene dimensiones menores que el video original y utiliza generalmente alguna de sus esquinas.

4. ***Encode:*** Reproduce los *artifacts* causados por las codificaciones continuas en videos. Explotan patrones de los videos en la dimensión temporal, tales como regiones similares

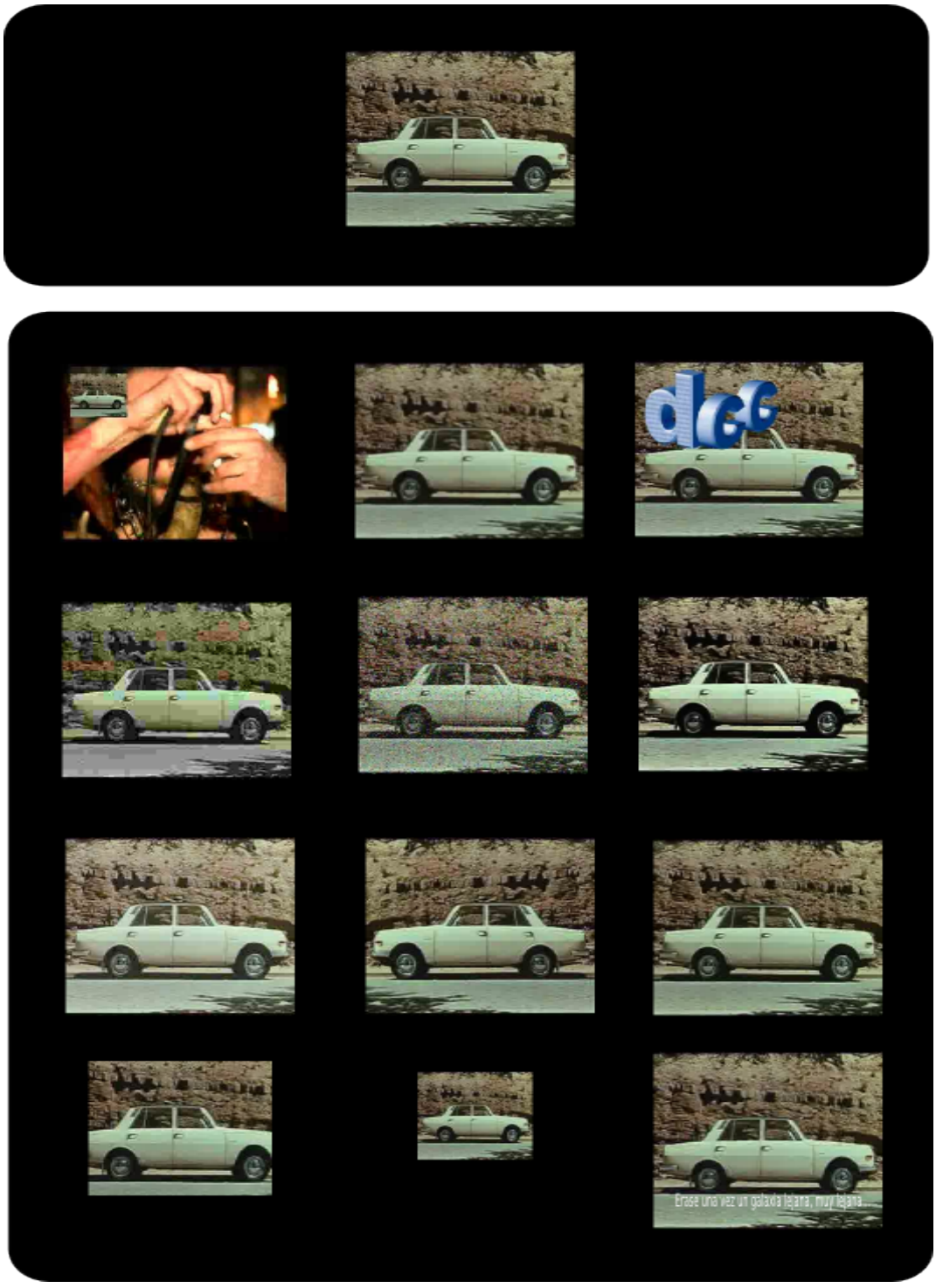

Figura 2.2: Doce transformaciones típicas sobre flujos de video en televisión, cine e Internet, para probar robustez en los descriptores. De izquierda a derecha, arriba hacia abajo: *Picture in picture*, *Blur*, *Captions*, *Encode*, *Noise add*, *Contrast*, *Gamma*, *Mirroring*, *Change in aspect ratio*, *Crop*, *Resize* y *Subtitles*.

en *frames* contiguos y *frames* repetidos, para reducir información en el video. También se utilizan codificaciones sólo sobre las imágenes, utilizando información espacial y de colores (generalmente promedios de pixeles vecinos).

5. ***Noise:*** Simula el ruido causado en televisiones con baja recepción de la señal. En este caso se disponen sobre la imagen pixeles de forma uniforme, con información de color aleatoria.

6. **Contraste:** Es un cambio en las diferencias de intensidades entre los elementos más brillantes y más oscuros de las imágenes o videos, generalmente causado por diferentes configuraciones en los dispositivos de salida. Una transformación de contraste se puede realizar mediante la normalización de la intensidad de los pixeles, sujeto a restricciones como las de la Ecuación 2.7 (siendo $I_{\max}$, $I_{\min}$ las intensidades máximas y mínimas y C un parámetro de ajuste) [13].

$$\frac{I_{\max} - I_{\min}}{I_{\max} + I_{\min}} = C \tag{2.7}$$

7. ***Change in gamma:*** Es una operación para variar la codificación y decodificación de la cantidad de luminosidad en la imagen, generalmente causado por diferentes configuraciones en los dispositivos de salida. Se habla de *Gamma correction*, puesto que se utiliza como transformación para corregir las diferencias entre distintos dispositivos. Para efectuar la corrección se utiliza la Ecuación 2.8, en donde las intensidades de entradas de los pixeles (I_{in}) se potencian por un exponente γ en la salida (I_{out}) [16].

$$I_{\text{out}} = {I_{\text{in}}}^{\gamma} \tag{2.8}$$

8. ***Vertical mirroring:*** Reflexión sobre la vertical en el video de entrada. Esta transformación es poco común y se asocia a copias ilegales de videos. La información de la imagen no se modifica ni en colores ni en intensidades, pero se produce un cambio espacial de la información. La transformación *Mirroring* para una imagen

$I(x, y,), \quad 1 \leq x \leq W, \quad 1 \leq y \leq H$ se puede escribir matemáticamente como en la Ecucación 2.9.

$$I(x, y) = I(W - x, y) \tag{2.9}$$

9. ***Change aspect ratio:*** Cambio en las proporciones del video. El aspecto de un video (o de una imagen), es la proporción entre las dimensiones espaciales del mismo ($\frac{\text{Width}}{\text{Height}}$) y no contiene cambios en las intensidades ni en los colores. Se asocia a diferencias en los dispositivos de salida.

10. ***Crop:*** Recorte en una o dos dimensiones de los videos. Esta transformación elimina columnas o filas de pixeles de las imágenes de los videos, pudiendo modificar no sólo la relación de aspecto sino también el tamaño espacial de los mismos.

11. **Inserción de texto:** Inserción de texto en la parte inferior de cada uno de los frames. Se asocia a inserción de subtítulos y explicaciones de los videos. Generalmente afectan una región acotada del video (por ejemplo $\frac{1}{6}$ de la región inferior).

Las transformaciones pueden agruparse en distintas categorías: algunas son *globales* -pues afectan toda la imagen- en cambio otras son *locales*. Algunas operan sobre el *espacio de colores* de la imagen y otras sobre el de *intensidades*. Algunas son *reversibles* mientras que otras no. También pueden ser *aditivas* (agregan información) o *substractivas* (eliminan información). En el Cuadro 2.1 se clasifican 12 transformaciones que se utilizaron en este trabajo (representadas en la Figura 2.2).

2.3. Descriptores

En general, una función de extracción de características recibe un objeto como entrada y devuelve una representación matemática del mismo en algún espacio vectorial. Se habla también de "huellas" o "vector característico", puesto que dado un objeto particular se le asigna un y sólo un valor del espacio vectorial. Dada una colección de objetos, se utiliza una función de extracción de características para generar los vectores asociados a cada objeto,

	Global/Local	**Espacio**	**Reversibles**	**Aditivas/Substractivas**
Picture in Picture	Local	Colores	Irreversible	Aditiva
Blur	Global	Colores	Irreversible	Substractiva
Captions	Local	Colores/Intensidades	Irreversible	Aditiva
Encode	Global	Colores/Intensidades	Irreversible	Aditiva
Noise	Global	Colores	Irreversible	Aditiva
Contrast	Global	Colores/Intensidades	Reversible/Irreversible	Aditiva/Sustractiva
Gamma	Global	Intensidades	Reversible	no aplica
Mirroring	Global	Cartesiano	Reversible	no aplica
Aspect ratio	Global	Cartesiano	Reversible	Aditiva/Sustractiva
Crop	Global/Local	Cartesiano	Irreversible	Sustractiva
Resize	Global	Cartesiano	Reversible	Aditiva/Sustractiva
Subtles	Local	Intensidades/Colores	Irreversible	Aditiva

Cuadro 2.1: Taxonomía de las transformaciones sobre video.

Figura 2.3: Tres transformaciones para un mismo *frame* de un video, pero con distintos parámetros. De arriba hacia abajo: *blur*, contraste y *Pattern*. De izquierda a derecha: parámetros suaves, medios y agresivos.

a fin de situarlos en un espacio vectorial en donde se conserven las similitudes inherentes a dichos objetos.

Las funciones de extracción de características también se aplican a imágenes; algunos tipos de extracción de características para imágenes son: histograma de colores, extracción de formas y bordes y puntos invariantes, entre otras [5] [18].

En cuanto a video, se basan principalmente en aplicar descriptores para imágenes a los *frames*.

2.3.1. Taxonomía de descriptores

En este trabajo, los descriptores se clasificarán por tipo de extracción de características:

1. **Locales**: Extraen una serie de puntos o regiones características de las imágenes y forman con ellas un vector de cierta dimensión. Por ejemplo, extraer los valores extremos de una transformada que se aplica sobre la imagen y almacenarlos en un vector [22].

2. **Globales**: Efectúan un resumen de toda la imagen para extraer un vector. Por ejemplo, crear un histograma de colores por cada *frame*, clasificando cada pixel en alguno de los *bins*, generando un vector con tantas dimensiones como *bins* se consideren en la clasificación [20].

También se clasificarán por la dimensión en la que se focalizan:

1. **Espaciales**: Se aplican las funciones de extracción de características a cada uno de los frames por separado, tratando los videos como un conjunto de imágenes. El método más simple es escoger un descriptor de imágenes n-dimensional y aplicarlo a cada uno de los d *frames* de un video. El descriptor del video queda determinado por una matriz de dimensiones $n \times d$.

2. **Temporales**: Extraen características utilizando la variable temporal del video. Para ello pueden hacer seguimiento temporal de puntos obtenidos con un descriptor local [12] [22]. La idea es que la extracción de características no considere un solo frame, sino que un sub-conjunto de ellos.

3. **Espacio-Temporales**: Son una combinación de los anteriores [23] [11].

Por último, se clasificarán de acuerdo a la forma de escoger los *frames* de un video sobre los que se aplicará la extracción de características; se utilizan actualmente alguna de las siguientes estrategias [10]:

1. ***Key-Frame-based***: Se extraen *frames* claves (*keys*) del video. Los *frames* claves son aquellos que representan a un sub-conjunto mayor de *frames*.

2. ***Group-of-frames***: Se extrae un grupo o grupos contiguos de *frames* del video.

3. ***Frame-by-Frame***: Se extraen todos los *frames* del video.

2.3.2. Propiedades de los descriptores

Las propiedades deseables para las implementaciones de los descriptores son básicamente: correctitud, robustez, eficiencia y eficacia. A continuación se definen cada una de estas propiedades:

Correctitud: Un descriptor se dice *correcto*, si en cada aplicación sobre un conjunto de datos se obtienen los resultados (en forma de vectores característicos) propuestos por sus autores. Esta definición es sólo para determinar si las implementaciones de esta memoria realmente coiciden con lo descrito por los autores en sus publicaciones.

Robustez: Un descriptor se dice *robusto*, si frente a cambios en las entradas (transformaciones sobre los *frames*) los resultados (en forma de vectores característicos) propuestos por los autores no varían mayormente. De este modo, un descriptor es más robusto, mientras la extracción de características sitúe los objetos en un espacio tal que se respeten las diferencias y similitudes visuales en y entre cada objeto.

La correctitud es más bien una característica asociada a una descripción detallada y fundamentada del mismo, a través de sus autores. La robustez está relacionada con la idea de similitud entre objetos: si dos objetos distintos comparten características, deberá reflejarse de una u otra forma en sus vectores característicos.

Eficiencia: La eficiencia de un descriptor se medirá de acuerdo a la cantidad de *frames* que procesa por unidad de tiempo (en este caso *frames*/s). Mientras mayor sea esta cantidad, más eficiente será el descriptor. Como "cero" se considera la reproducción normal del video (generalmente 24 fps o 30 fps).

Eficacia: Un descriptor es más eficaz mientras más resultados relevantes entregue del total de relevantes que existe en una colección y mientras más resultados relevantes contenga el conjunto de resultados.

2.4. Búsqueda por similitud

En búsqueda por similitud no se pretende encontrar un objeto en particular, sino un subconjunto de objetos de la colección sobre la que se busca, en donde todos los resultados guarden algún grado de similitud respecto al objecto de consulta. Actualmente existen varias definiciones posibles para este término, por ejemplo, dos objetos pueden considerarse similares:

1. Mientras más cerca estén, considerando una función de distancia dada en el espacio vectorial de los descriptores (caso vectorial).

2. Cuando el costo de transformación entre ellos es mínimo (caso no vectorial). El costo de transformación se obtiene asociando valores a transformaciones sobre un objeto. El costo total es la suma de cada uno de los costos realizados para transformar un objeto en otro.

3. Existen partes concordantes entre ellos (caso no vectorial). Cuando los objetos se basan en componentes, se puede definir similitud basándose en la cantidad de componentes en posiciones iguales entre los objetos.

2.5. Tipos de búsqueda

Existen tres tipos básicos de consultas por similitud:

1. **Búsqueda por rango:** Sea $\mathbb{U}$ el universo de objetos, $q \in \mathbb{U}$ el objeto consulta, $\mathbb{S}$ la colección de datos donde se está buscando tal que $\mathbb{S} \subset \mathbb{U}$ y $r \in \mathbb{R}^+$ el radio de tolerancia; se pide entonces un conjunto (q, r) (Ecuación 2.10).

$$(q, r) = \{s \in \mathbb{S}, \delta(s, q) \leq r\} \tag{2.10}$$

La búsqueda por rango retorna todos los objetos del universo de búsqueda $\mathbb{U}$ que están a lo más a cierta distancia dada del objeto q.

2. **K-Vecinos Más Cercanos (K-NN: *K-nearest neighbors*):** Sea $\mathbb{U}$ el universo de objetos, $q \in \mathbb{U}$ el objeto consulta y $k \in \mathbb{N}$. Se pide encontrar un subconjunto $\mathbb{K} \in \mathbb{S}$, tal que cumpla con la Ecuación 2.11.

$$\forall x \in \mathbb{K}, y \in \mathbb{S} \setminus \mathbb{K}, \delta(x, q) \leq \delta(y, q). \tag{2.11}$$

K-NN encuentra los k objetos más cercanos al objeto de consulta q en el universo $\mathbb{U}$.

3. **Búsqueda incremental:** En este caso el usuario no conoce un radio de tolerancia r adecuado ni tampoco la cantidad de elementos k más cercanos. Se realiza una búsqueda K-NN utilizando valores incrementales de k. El usuario especifica un k inicial que se incrementa en cada operación de búsqueda para obtener más resultados.

2.6. Espacios vectoriales y métricos

Sea $\mathbb{U}$ el conjunto universal de objetos válidos, si a cada objeto $u \in \mathbb{U}$ aplicamos un descriptor n-dimensional, obtenemos una n-tupla por cada uno de ellos. Si cada componente de una n-tupla pertenece a algún "campo" [1] y definimos suma y multiplicación, entonces se tiene un espacio vectorial. Formalmente se escribe como en la Ecuación 2.12 y 2.13

$$\mathbb{Z} := \{\{.. - 2, -1, 0, 1, 2..\}, \oplus, \otimes\} \tag{2.12}$$

$$x \in \mathbb{Z}^n \Rightarrow x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \tag{2.13}$$

[1] Estructura algebraica donde se han definido las cuatro operaciones básicas: suma ($\oplus$), resta ($\ominus$), multiplicación ($\otimes$), división ($\oslash$) y se cumplen los axiomas de clausura, asociatividad, conmutatividad, inverso multiplicativo/aditivo, identidad suma/multiplicación y distributividad. Por ejemplo $\mathbb{R}, \mathbb{Z}$ y $\{0, 1\}$

2.6.1. Funciones de distancia

Para estos espacios vectoriales, se puede definir una función de distancia δ tal que recibe dos objetos del conjunto y devuelve un número real no negativo (ver Ecuación 2.14).

$$\delta : \mathbb{U} \times \mathbb{U} \to \mathbb{R}^+ \tag{2.14}$$

Algunas funciones de distancia para estos espacios vectoriales son:

1. **Minkowski**: $\delta(\vec{x}, \vec{y}) = (\sum_{i=1}^{n} \|x_i - y_i\|^p)^{1/p}$. Donde p es un valor fijo, $p \geq 1$

   a) Si $p = 1$ se obtiene la distancia Manhattan.

   b) Si $p = 2$ se obtiene la distancia Euclidiana.

   c) si $p \to \infty, d(\vec{x}, \vec{y}) = \text{máx}_{i=0}^{n} \|x_i - y_i\|$.

2. **Mahalanobis**: $\delta(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T S^{-1} (\vec{x} - \vec{y})}$. Donde x e y son vectores de una misma distribución con matriz de covarianza S.

3. **Hamming**: En un espacio vectorial binario (donde los componentes de cada tupla pertenecen a $\{0, 1\}$), la distancia Hamming calcula la cantidad de cambios necesarios en los componentes de una de las tuplas para transformarla en otra.

2.6.2. Espacio métrico

La tupla $(\delta, \mathbb{U})$ recibe el nombre de **espacio métrico** si y sólo si δ cumple con las siguientes propiedades:

1. Positividad estricta: $\forall x, y \in \mathbb{U}, x \neq y \Rightarrow \delta(x, y) > 0$

2. Simetría: $\forall x, y \in \mathbb{U}, \delta(x, y) = \delta(y, x)$

3. Reflexividad: $\forall x \in \mathbb{U}, \delta(x, x) = 0$

4. Desigualdad triangular: $\forall x, y, z \in \mathbb{U}, \delta(x, z) \leqslant \delta(x, y) + \delta(y, z)$

La primera propiedad obliga a que toda función de distancia en un espacio métrico debe ser estrictamente positiva para todo par de objetos distinto en el universo comprendido. La segunda establece indiferencia en el orden de los objetos comparados. La tercera indica que la distancia de un objeto a sí mismo es cero (mínima distancia). Por último, la cuarta propiedad establece que: siempre se obtiene menor (o igual) distancia al comparar un objeto con otro, que comparando mediante objetos intermedios.

2.7. Búsqueda en video

Dentro de cada uno de los tipos de búsqueda de la Sección 2.5 se comparan descriptores mediante funciones de distancia como en la Sección 2.6.1. Pero en general para un descriptor dado, los vectores característicos de dos objetos, pertenecen a distintos espacios. Por ejemplo, dados dos videos V_1, V_2, de largos n, m respectivamente, un descriptor puede devolver dos secuencias de vectores $\mathbb{D}_1, \mathbb{D}_2$ tales que $\|\mathbb{D}_1\| = 1.000$ y $\|\mathbb{D}_2\| = 5.000$ con cada vector en $\mathbb{R}^n$ y $n = 64$. De esta forma, al comparar descriptores de distintos videos, se están comparando dos matrices (siendo cada columna un vector característico de la secuencia) de distintas dimensiones. Para esto, algunos autores [11] [14] proponen un algoritmo de "búsqueda exhaustiva" mostrado a continuación:

Algoritmo Búsqueda exhaustiva en video

Básicamente la búsqueda exhaustiva recorre el video más largo, comparando contra los *frames* del video más corto avanzando de un frame (V_1^j indica el *frame* j-ésimo del video V_1). Al finalizar el algoritmo devuelve la distancia mínima encontrada entre los *frames* del video más corto y una secuencia consecutiva de *frames* del video más largo.

2.8. Evaluación de resultados en búsquedas por similitud

En este trabajo se utilizará la siguiente definición formal para evaluar eficacia (o efectividad):

Algorithm 1 Algoritmo de búsqueda exhaustiva para videos

$m := min(\|V_1\|, \|V_2\|)$
$M := max(\|V_1\|, \|V_2\|)$
$r := M - m$
$minDist := \infty$
$V_1 := Argmin(\|V_1\|, \|V_2\|)$
$V_2 := Argmax(\|V_1\|, \|V_2\|)$
for $i := 0$ to $i := M$ **do**
  $dist := 0, 0$
  for $j := 1$ to $j := m$ **do**
    $dist := dist + d(V_1^j, V_2^{j+i})$
  end for
  if $dist < minDist$ **then**
    $minDist := dist$
  end if
end for
return minDist

Eficacia: es la calidad de la respuesta percibida por el usuario; esto es, medir la capacidad de una búsqueda de recuperar objetos realmente relevantes.

Con esto se pueden definir los siguientes valores:

	Relevante	**No relevante**
Encontrado	Positivo Correcto (RP)	Falso Positivo (FP)
No Encontrado	Falso Negativo (FN)	Negativo Correcto (RN)

Teniendo esta clasificación en mente, se define la siguiente medida de efectividad:

1. ***Precision-Recall*** [2]. *Precision* se define como la cantidad de objetos que son relevantes en un conjunto de resultados. *Recall* se define como la cantidad de objetos recuperados del total de relevantes. En términos formales se escriben de acuerdo a las Ecuaciones 2.15 y 2.16.

$$\textbf{Precision} := \frac{RP}{RP + FP} \tag{2.15}$$

$$\textbf{Recall} := \frac{RP}{RP + FN} \tag{2.16}$$

La medida de *Precision-Recall* se grafica en los ejes X para *Recall* e Y para *Precision*. La Figura 2.4 muestra un ejemplo de un gráfico *Precision vs Recall*: el caso ideal ocurre cuando para distintas densidades de objetos relevantes recuperados (*Recall* en el intervalo $[0, 1]$), se obtienen sólo objetos relevantes en el conjunto de resultados ($Precision = 1.0$). En este caso al realizar una búsqueda, todos los objetos obtenidos en el conjunto de resultados son relevantes; sin importar el tamaño del conjunto de objetos relevantes ni el tamaño del conjunto de resultados, siempre se podrán obtener todos los objetos relevantes.

La mayoría de los sistemas de búsqueda tienden a tener curvas como "Ejemplo" en la Figura 2.4. Esto pues en la mayoría de los sistemas de búsqueda, para aumentar la proporción de objetos relevantes encontrados del total de relevantes de la colección, se debe aumentar el tamaño del conjunto de resultados, cambiando la fracción de objetos relevantes encontrados del total de objetos obtenidos en la búsqueda. Sin embargo, un aumento en la proporción de objetos relevantes encontrados del total de relevantes de la colección (aumento en *Recall*), incide en que muchos de los nuevos resultados encontrados no sean relevantes (haciendo caer *Precision*).

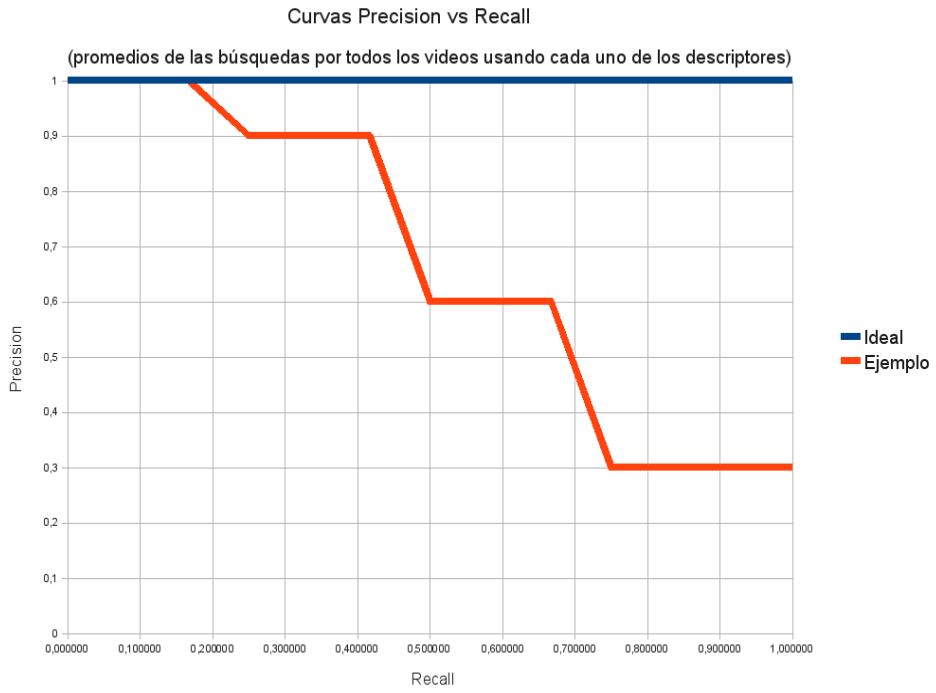

Figura 2.4: Gráfico ejemplo con curvas de precision-recall ("Ideal" y un descriptor "Ejemplo" promedio).

Capítulo 3

Descriptores

En esta memoria se implementan los siguientes descriptores:

1. **Edge-Desctiptor**. Iwamoto *et al. Image Signature Robust to Caption Superimposition for Video Sequence Identification* [10].

2. **Mean-Average Descriptor**. Kim *et al. Spatio Temporal Sequence Matching for Efficient Video Copy Detection* [11].

3. **BCS descriptor**. Shen *et al. UQLIPS: A Real Time Near-Duplicate Video Clip Detection System* [17].

4. **DCT descriptor**. Naturel *et al. A Fast Shot Matching Strategy for Detecting Duplicate Sequences in a Television Stream* [14].

3.1. *Edge descriptor*

Descriptor del tipo global, espacial y *frame-by-frame*. Pretende ser de bajo costo computacional y robusto a *captions*[1]. Al ser robusto a *captions*, permite identificar videos idénticos pero que han sido estampados con algún logo o subtítulos. Opera en un espacio de colores basado en intensidades (en este caso YCbCR). Se compone de los siguientes pasos:

1. **Extracción de características:** Se recorre el video *frame* por *frame* (frame-by-frame) y en cada uno de ellos:

[1] títulos, subtítulos o leyendas; muy típicos en programas de televisión por ejemplo

-1.0	1.0
-1.0	1.0

0.0	$\sqrt{2}$
$-\sqrt{2}$	0.0

1.0	1.0
-1.0	-1.0

$\sqrt{2}$	0.0
0.0	$-\sqrt{2.0}$

1.0	-1.0
1.0	-1.0

0.0	$-\sqrt{2}$
$\sqrt{2}$	0.0

-1.0	-1.0
1.0	1.0

$-\sqrt{2}$	0.0
0.0	$-\sqrt{2}$

2.0	-2.0
-2.0	2.0

-2.0	2.0
2.0	-2.0

Cuadro 3.1: Diez *kernels* de borde para "Edge-Descriptor". De izquierda a derecha, partiendo desde arriba: $0, 45, 90, 135, 180, -135, -90, -45$ grados y No-Direccional 1 y No-Direccional 2.

a) Se divide el *frame* en K^2 bloques, donde K es un parámetro dado.

b) Cada bloque es particionado en 4 sub-bloques.

c) Por cada sub-bloque se calcula la intensidad promedio de los pixeles que contiene.

d) A cada uno de los K^2 bloques se le aplican los 10 filtros de convolución del Cuadro 3.1, para detectar los gradientes de intensidad.

e) Por cada bloque, se busca el filtro que obtuvo el mayor valor. Si no supera cierto valor umbral T (*threshold*) -dado como parámetro- entonces se clasifica con un valor numérico correspondiente a un bloque "sin borde". En caso contrario se marca ese bloque con un indicador numérico único para el filtro.

El descriptor es una matriz, siendo cada una de sus columnas el vector característico de cada *frame*. Cada vector sera uno K^2-dimensional (una dimensión por cada bloque del frame, con un número que identifica de forma única al filtro para ese bloque) y para un video de largo n se tendrán entonces nK^2 vectores almacenados secuencialmente en una matriz. El resultado del descriptor se aprecia en la Figura 3.1.

2. **Calce de características:** Sean $\vec{X}$ e $\vec{Y}$ los vectores característicos de dos *frames* pertenecientes al video consulta y al video consultado respectivamente. Se define una función de coincidencias $s(x, y)$ en las Ecuaciones 3.1 y 3.2.

$$s(\vec{x}, \vec{y}) = \sum_{i=1}^{N} E(x_i, y_i) \tag{3.1}$$

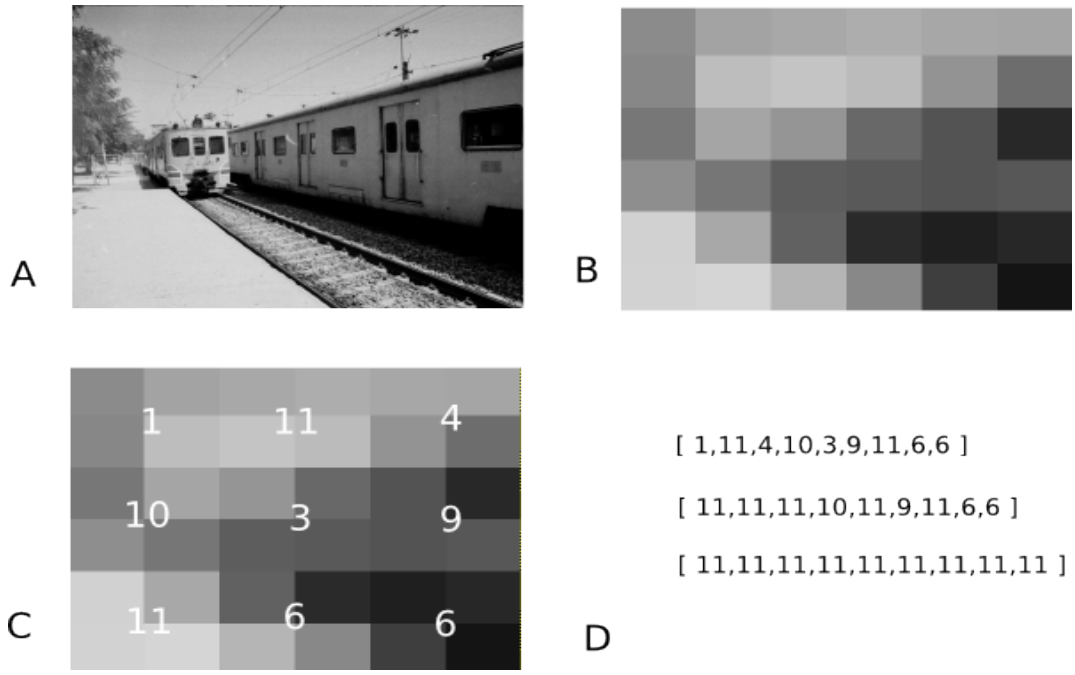

Figura 3.1: A: Frame perteneciente a un video cualquiera. B: El mismo *frame* dividido 3×3 bloques, cada uno con sus 2×2 sub-bloques e intensidades promedios. C: Superposición de B con el resultado de clasificación del descriptor utilizando como parámetro $T := 20$. D: Vector descriptor del *frame* para $T := \{20, 100, 200\}$.

$$E(x_i, y_i) = \begin{cases} 1 & \text{si} \quad x_i = y_i \\ 0 & \text{si} \quad x_i \neq y_i \end{cases} \tag{3.2}$$

Esta función devuelve información de las veces que dos bloques, en una misma posición de dos *frames* pertenecientes a dos videos distintos, fueron clasificados con el mismo número.

3.2. *Mean average descriptor*

Descriptor del tipo global, espacio-temporal y *frame-by-frame*. Pretende ser robusto a distorsiones (efecto de la codificación del video) y cambios de formato. Este descriptor opera en espacio de colores de intensidades (YCbCr). Se efectúan los siguientes pasos (ver Figura 3.2):

1. **Extracción de características:**

   a) Cada *frame* se particiona en K^2 bloques, donde K es parámetro dado.

   b) Se calcula la intensidad promedio de los pixeles contenidos en cada bloque.

c) Los bloques se numeran de acuerdo a sus intensidades con números de $[1, ..., K^2]$, en donde 1 corresponde al bloque con mayor intensidad y K^2 al de menor.

d) Se obtiene una matriz $K \times K$ (matriz de *ranking* de los bloques del frame), en donde cada dimensión corresponde a un bloque de la imagen. Para los siguientes pasos de este descriptor, la matriz de *ranking* se transformará en un vector K^2-dimensional.

2. **Calce de características:**

a) Dado el video de consulta π_q y el objetivo π_t. Se define el *frame* i-ésimo para π_q como $\pi_{q,i}$ y el bloque j-ésimo del *frame* i-ésimo del video π_q como $\pi^j_{q,i}$. La función de distancia espacial que compara dos *frames* es d_e en Ecuación 3.3.

$$d_e(\pi_{q,i}, \pi_{t,i}) = \frac{1}{C} \sum_{j=1}^{K^2} \|\pi^j_{q,i} - \pi^j_{t,i}\| \tag{3.3}$$

Esto es, la diferencia en el *ranking* de cada bloque, de cada uno de los *frames* de los videos comparados. C es la máxima diferencia entre las dos matrices ordenadas que representan a los *frames* de cada video (factor de normalización). La función de distancia espacial que compara dos videos V_q y V_{qt} de largo N se define como D_E en la Ecuación 3.4.

$$D_E(V_q, V_t[p]) = \frac{\sum_{i=0}^{N-1} d_e(\pi_{q,i}, \pi_{t,p+i})}{N} \tag{3.4}$$

Donde $V_t[p]$ es un video objetivo que está inicialmente en el *frame* p. Notar que dicho video debe contener la misma cantidad N de *frames* que el video de consulta V_q, en caso contrario se utiliza la búsqueda exhaustiva de la Sección 2.7.

b) Para calcular la distancia temporal entre dos videos V_q y V_t, se usa $D_T(V_q, V_t[p])$ definida en las Ecuaciones 3.5 a 3.8.

$$D_T(V_q, V_t[p]) = d_t(\delta_q, \delta_t) \tag{3.5}$$

$$d_t(\delta_q, \delta_t) = \frac{1}{4(N-1)} \sum_{j=1}^{K^2} \sum_{i=1}^{N-1} f(\delta_{q,i}^j - \delta_{t,p+i}^j) \tag{3.6}$$

Donde se ha definido

$$f(x) = \frac{|x|}{2} \tag{3.7}$$

$$\delta_{q,i}^j = \begin{cases} 1, & \text{si} \quad V_{q,i}^j > V_{q,i-1}^j \\ 0, & \text{si} \quad V_{q,i}^j = V_{q,i-1}^j \\ -1, & \text{si} \quad V_{q,i}^j < V_{q,i-1}^j \end{cases} \tag{3.8}$$

Donde $V_{q,i}^j$ indica el bloque j-ésimo del *frame* i-ésimo para el video q.

Intuitivamente, $\delta_{q,i}^j$ indica un cambio temporal en el *ranking* del bloque j-ésimo en el video q, considerando el *frame* i-ésimo y el $(i-1)$-ésimo. De esta forma, la función $d_t(\delta_q, \delta_t)$ indica las diferencias en los cambios temporales entre los videos q y t.

c) Finalmente la función de distancia temporal y espacial viene dada por $D(V_q, V_t[p])$ en la Ecuación 3.9.

$$D(V_q, V_t[p]) = \alpha D_S(V_q, V_t[p]) + (1-\alpha) D_T(V_q, V_t[p]) \tag{3.9}$$

Donde α es un parámetro de ajuste para dar mayor énfasis a la búsqueda temporal o la búsqueda espacial.

3.3. *BCS descriptor*

Es un descriptor global, espacio-temporal y *frame-by-frame*. Pretende ser altamente eficiente y robusto a transformaciones locales y globales.

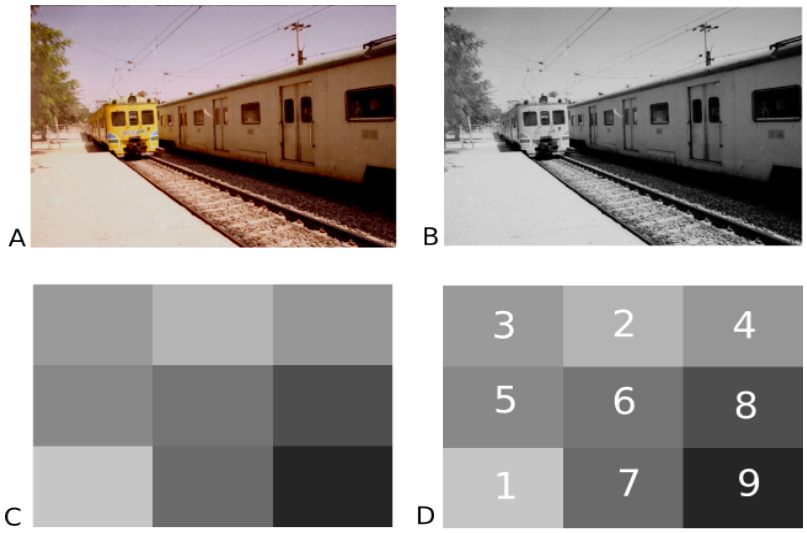

Figura 3.2: A: *Frame* perteneciente a un video cualquiera. B: *Frame* en espacio de intensidades. C: 3×3 bloques y sus respectivos promedios de intensidades. D: 3×3 bloques, sus intensidades promedios y la respectiva tabla de orden. E: Tabla de orden en forma vectorial.

1. **Extracción de características:**

   a) Aplicar D a cada uno de los n *frames* de V_q. Se obtiene un vector $X = \{\vec{x}_1, ..., \vec{x}_n\}$ con $\vec{x}_i$ vector d-dimensional $\forall i = 1...n$.

   b) Generar los vectores $BCS(X) = \left(O^X, \Phi_1^X, ..., \Phi_d^X\right)$. Donde O^X es la media de los vectores de X, y Φ_j^X es el i-ésimo *Bounded Principal Component* del *Bounded Coordinate System*, ambos explicados en la Sección 3.3.1.

2. **Calce de características:**

   La distancia entre dos descriptores BCS viene dada por $D(BCS(X), BCS(Y))$ (Ecuación 3.10)

$$D(BCS(X), BCS(Y)) = \|O_i^X - O_i^Y\| + \sum_{i=1}^{d} \|\Phi_i^X - \Phi_i^Y\| \tag{3.10}$$

3.3.1. Cálculo del *Bounded Coordinate System*

El *Bounded Coordinate System* (BCS) se utiliza para reducir la cantidad de datos que describirán a un video. Dado que el representar un video usando un descriptor de imagen por cada *frame* del mismo resulta en una gran cantidad de vectores, los autores proponen un sistema que captura la mayor cantidad de información reduciendo el número de descriptores por frame; para ello (ver Figura 3.3):

1. Se crea una matriz M que tiene en cada columna un vector d-dimensional correspondiente a uno de los N *frames* del video. De esta forma, se tendrá una matriz $N \times d$.

2. Se hace un análisis de componentes principales (PCA), teniendo como datos cada una de las columnas de la matriz. El análisis consiste en:

   a) Se calcula el origen O a partir de la media en cada una de las d-dimensiones de los datos en la matriz M.

   b) Normalizar cada dimensión de los vectores respecto a la media, obteniendo la matriz B.

   c) Calcular la matriz de covarianzas $C = BB^T$.

   d) Calcular los vectores y valores propios de la matriz C.

   Los vectores propios de C corresponden a los componentes principales de la matriz M. Cada componente principal corresponde a un eje respecto a la media de todos los vectores, que acumula cierta energía proporcional a la varianza de los datos en ese eje. *BCS* no es más que un *PCA* donde se han considerado todos los componentes principales, sin importar la energía que acumulen (en *PCA* se extraen sólo los de mayor energía).

3. Por cada uno de los componentes principales, se proyectan los datos y se calcula la desviación estandar σ_i respecto al origen O.

4. Cada uno de los componentes principales se escala por la desviación estándar correspondiente.

Cada uno de los d vectores obtenidos, corresponde a un *Bounded Principal Component* y en conjunto conforman un *Bounded Coordinate System*. Intuitivamente, un BCS no es más que una reducción en la cantidad de datos (descriptores) que componen a un video, a modo de agilizar los cálculos de distancias.

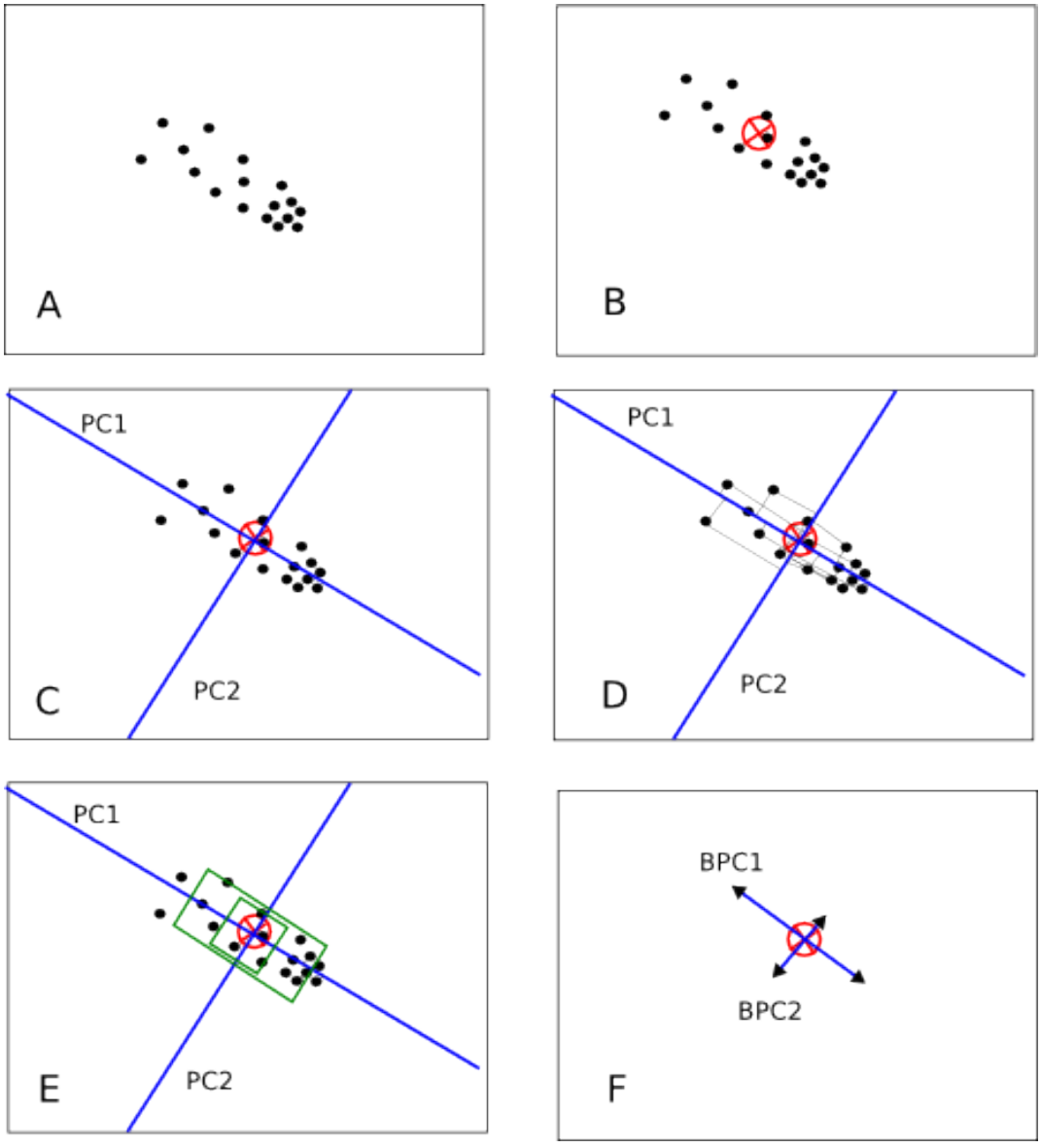

Figura 3.3: A: Descriptores de 2 dimensiones para un video cualquiera. B: Cálculo del centro de masa o media de los descriptores. C: Cálculo de los componentes principales. D: Proyección de los datos sobre los componentes principales. E: Cálculo de las desviaciones estándar σ_i. E y F: Cálculo de los *Bounded Principal Component*.

3.4. *DCT based descriptor*

Es un descriptor del tipo global, espacial y *frame-by-frame*. Se especializa en búsqueda de *shots* repetidas en videos. Robusto a ruidos de *broadcasting* (ruido gaussiano, cambios de colores y artifacts[2] debido a compresiones). Dado un video de consulta V_q se pide (ver Figura 3.4 como apoyo):

1. **Extracción de características:**

   a) A cada *frame* se le aplica la transformada coseno discreta (DCT). Dado un *frame* de dimensiones $N \times M$, determinado por la función $I(x, y)$ —que representa la intensidad del pixel i, j en el *frame*—, se calcula la DCT mediante la Ecuación 3.11.

$$DCT(u,v) = \alpha(u,v) \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} cos\left[(2x+1)\frac{\pi u}{2MN}\right] cos\left[(2y+1)\frac{\pi v}{2M}\right] I(x,y) \tag{3.11}$$

$$u, v \in \mathbb{Z}$$
$$0 \leq u \leq N, 0 \leq v \leq M$$

   Siendo $\alpha(u,v) = \frac{2}{\sqrt{MN}} C(u)C(v)$ con $C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{para } u = 0 \\ 1 & \text{caso contrario} \end{cases}$.

   Se extraen los coeficientes más significativos de la matriz $DCT(u, v)$. Los coeficientes más significativos vienen dados por las frecuencias bajas de la matriz, esto es los coeficientes tales que $0 \leq u \leq n$ y $0 \leq v \leq n$ siendo $5 \leq n \leq 8$ un parámetro experimental. Las frecuencias altas (coeficientes menos significativos) vienen dados por el cuadrante inferior derecho de la matriz. Los coeficientes significativos son cuantizados mediante la Ecuación 3.12.

$$\sigma(i) = \begin{cases} 1 & \text{si } DCT(\lfloor \frac{i}{n} \rfloor, i - \lfloor \frac{i}{n} \rfloor n) \geq m \\ 0 & \text{caso contrario} \end{cases} \quad \text{para } i \in [1, n^2] \tag{3.12}$$

[2]Pérdida de calidad en videos debido a la compresión

Donde m es la media de los primeros n^2 coeficientes de la matriz.

Notar que la DCT son una serie de coeficientes que representan la información presente en la función de entrada $I(x, y)$. Por ejemplo, el coeficiente $DCT(0, 0)$ corresponde a la media de la función de entrada. Esto se verifica haciendo $u = v = 0$, con esto los dos términos $cos[...]$ se hacen uno, dejando la Ecuación 3.13.

$$DCT(u, v) = \frac{2}{\sqrt{2NM}} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} I(x, y) \tag{3.13}$$

Notar que el factor $\alpha(u, v)$ es para evitar el factor de conversión para obtener la antitransformada de la DCT.

Cada *frame* entonces se representa con un vector binario de dimensión n^2 donde $5 \leq n \leq 8$. Por tanto la dimensión d de los vectores cumple con $25 \leq d \leq 64$ siendo posible el mapeo de este vector a un entero de 64-bit.

2. **Calce de características:**

   a) Se calcula la distancia entre dos videos v_q y v_i en alguna colección de videos mediante $D_{N\alpha}(v_q, v_i)$ (Ecuación 3.14)).

$$D_{N\alpha}(v_q, v_i) = \frac{1}{N} \sum_{i=1}^{N} d_h(\sigma_{q_i}, \sigma_{p_i}) \tag{3.14}$$

   Donde $v_q = \{\sigma_{q_1}, \sigma_{q_2}, ..., \sigma_{q_N}\}$ y $v_i = \{\sigma_{p_1}, \sigma_{p_2}, ..., \sigma_{p_N}\}$ corresponden a los vectores característico de cada una de los *frames*. La distancia d_h corresponde a la distancia Hamming descrita en la Sección 2.6.

   b) Por cada video en el que se está buscando un calce, se utiliza un algoritmo como el descrito en la Sección 2.7.

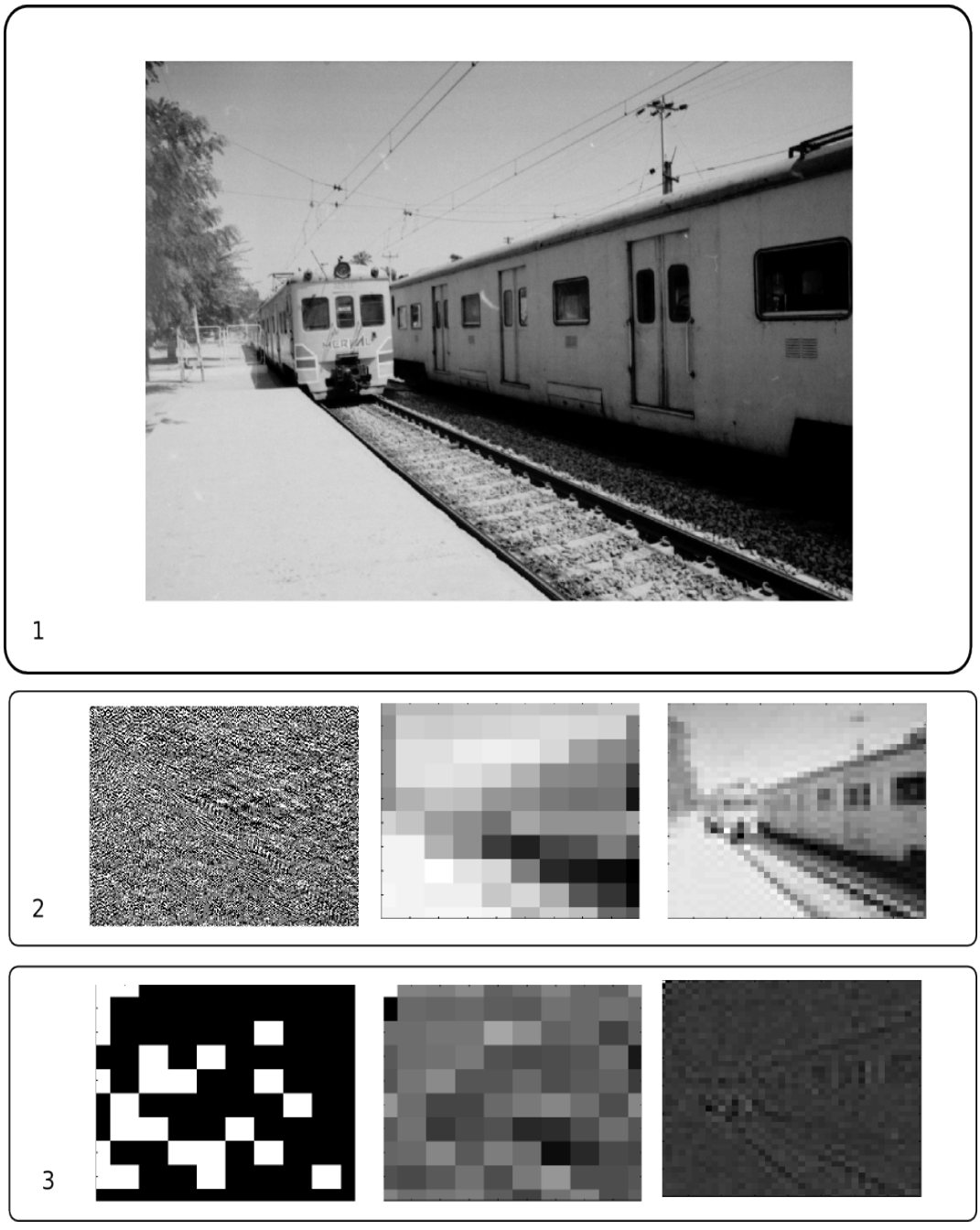

Figura 3.4: Recuadro 1: *Frame* de un video cualquiera. Recuadro 2 (izq. a der.): Todos los coeficientes de la transformada coseno discreta para el *frame* en 1. Inversa de la transformada coseno discreta de los 10×10 primeros coeficientes para el *frame* en 1. Inversa de la transformada coseno discreta de los 80×80 primeros coeficientes para el *frame* en 1. Recuadro 3: 10×10 primeros coeficientes de la transformada coseno discreta cuantizada para el *frame* en 1. Inversa de la transformada coseno discreta cuantizada de los 10×10 primeros coeficientes para el *frame* en 1. Inversa de la transformada coseno discreta cuantizada de los 80×80 primeros coeficientes para el *frame* en 1.

Capítulo 4

Implementación

4.1. Propósito

La implementación tiene como objetivo definir una serie de programas que permitan comparar los descriptores en cuanto a eficacia y eficiencia, además de servir de base para futuras investigaciones que utilicen descriptores sobre videos y/o imágenes. Para ello la implementación realizada permite las siguientes funcionalidades:

1. Cargar uno o varios videos desde un directorio -y subdirectorios- y extraer sus *frames* a una carpeta destino mediante un prefijo.

2. Aplicar uno o varios métodos de extracción de características (*Feature extraction*) a todos los *frames* en forma de imágenes dentro de un directorio, guardar el o los vectores característicos en un archivo y calcular la distancia a otro descriptor seleccionado.

3. Cargar el o los vectores característicos de un video desde un archivo.

4. Buscar los videos similares a un video de consulta, dentro de un directorio que contiene archivos de descriptores y/o sub-directorios con archivos de descriptores.

4.2. Uso

4.2.1. Usuarios

1. **Usuario experimental:** Compara descriptores de videos mediante distancias.

4.2.2. Casos de uso

A continuación se detallan los casos de uso propuestos para los programas junto con un esquema de los mismos (Figura 4.1).

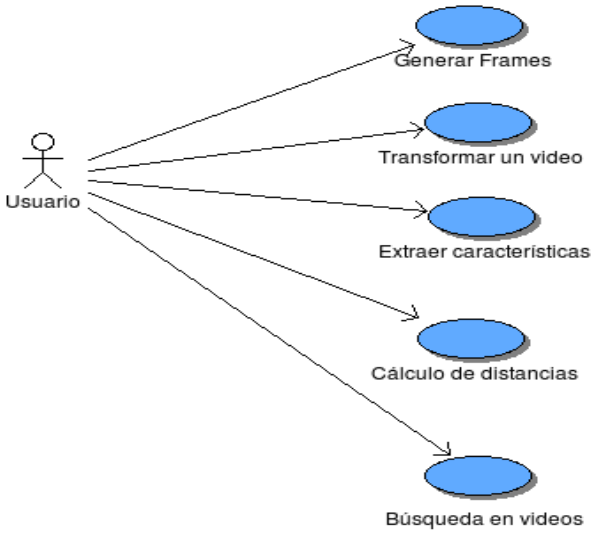

Figura 4.1: Casos de uso propuestos para la implementación.

1. **Generar *frames*:** El usuario ingresa un directorio base, un tipo de extensión de video (formato en el que están guardados los videos) y un prefijo para nombres de archivos. El programa busca por videos con el formato indicado, en cada sub-directorio del directorio base. Por cada video, genera una imagen para cada uno de los *frames* del video utilizando el prefijo de nombres de archivo. Se entiende que cada sub-directorio del directorio base contiene un y sólo un video.

2. **Transformar un video:** El usuario ingresa un directorio que contiene los *frames* de un video, su extensión, la transformación a aplicar y el directorio de salida de las imágenes. El programa arroja en el directorio de salida, las imágenes que contiene el directorio de entrada con la transformación aplicada, utilizando el mismo nombre de los archivos de entrada como prefijo.

3. **Extraer características**: El usuario ingresa un directorio (directorio base), un tipo de extensión de imágenes (formato en el que están guardados los frames), y uno o varios

métodos de extracción de características a aplicar con sus parámetros. El programa aplica el método de extracción de características con los parámetros entregados a cada una de las imágenes (*frames*) de cada uno de los directorios y sub-directorios. Se entiende que cada sub-directorio dentro del directorio base pertenece a un video en particular, y por ende debe contener imágenes (*frames*) en el formato especificado. Al final de la ejecución, cada directorio que corresponde a un video tendrá un archivo asociado al descriptor con un nombre único.

4. **Cálculo de distancias:** El usuario ingresa dos archivos correspondientes a descriptores de videos. El programa retorna la distancia entre los dos descriptores, utilizando la función de distancia asociada a cada descriptor.

5. **Búsqueda en videos**: El usuario ingresa un directorio donde se alojan subdirectorios, cada uno conteniendo los vectores característicos de un video, selecciona un descriptor y el programa devuelve la distancia de cada uno de los videos a todos los otros mediante la búsqueda exhaustiva descrita en la Sección 2.7.

4.3. Aspectos técnicos

Todas las clases se escribieron en C++. Este lenguaje provee manejo a nivel medio de los datos (por ejemplo mediante manipulación de bits e indirección) y por lo tanto ofrece mayor flexibilidad y eficiencia en los cálculos. Además soporta programación orientada a objetos [19].

Para todos los casos en que se usaron librerías externas, se utilizó una clase abstracta que define las operaciones que serán utilizadas. Esto permite que las clases no queden sujetas a un tipo de implementación de forma estricta.

El manejo de imágenes y video se hizo mediante la librería *OpenCV* [1]. La librería *OpenCV* está ampliamente documentada, es *open source*, hace uso de instrucciones extendidas en procesadores Intel (como SSE, SSE2, SSE3 e IPP) y existe un libro de referencia -escrito por los autores de la librería- que se usó en esta memoria como apoyo [3].

Para el manejo de directorios e iteradores sobre el sistema de archivos, se utilizaron las

[1] http://opencv.willowgarage.com/

librerías provistas por *Boost* [2], por ser *peer-reviewed*, de uso libre y con extensa documentación existente.

Para la diagramación de clases pre y post implementación se utilizó BoUml [3].

4.4. Diagrama de clases

A continuación se detallan las clases implementadas junto con diagramas de clases. El diagrama completo de clases se muestra en la Figura 6.1 del Apéndice.

4.4.1. Clases: *Image* y *Video*

Las clases *Image* y *Video* (ver Figura 4.2) se proveen como puente para manejar imágenes y videos. Ambas son clases puramente abstractas y sirven para definir la estructura de las funciones y métodos que usarán las clases que se sirvan de ellas. Para la implementación concreta de ambas clases se utilizó la librería de Intel *Open CV*.

La clase *Image* tiene métodos que permiten obtener la información de un pixel y modificarlo (colores e intensidades), obtener el promedio de intensidades o de color de una región rectangular de la imagen, obtener el tamaño, guardar la imagen en un archivo y mostrar la imagen en pantalla. Esta clase se basa en el patrón de diseño *Bridge* [7].

La clase *Video* tiene un objeto del tipo *VideoIterator*, el cual permite obtener los *frames* de un video uno por uno o alguno en particular, en una instancia de la clase *Image*. Esta clase se basa en el patrón de diseño *Iterator* y *Bridge* [7].

La clase *CVVideoIterator* implementa la clase *VideoIterator* utilizando la librería *OpenCV*. Entrega un frame como objeto *cvImage*. Se provee para extraer los *frames* de un video y guardarlos en disco. Para ello se crea un objeto del tipo *CVVideo*, se obtiene el iterador correspondiente, se itera sobre el video y a cada objeto *cvImage* obtenido se le aplica el método para guardar a disco.

[2] http://www.boost.org/
[3] http://bouml.sourceforge.net/

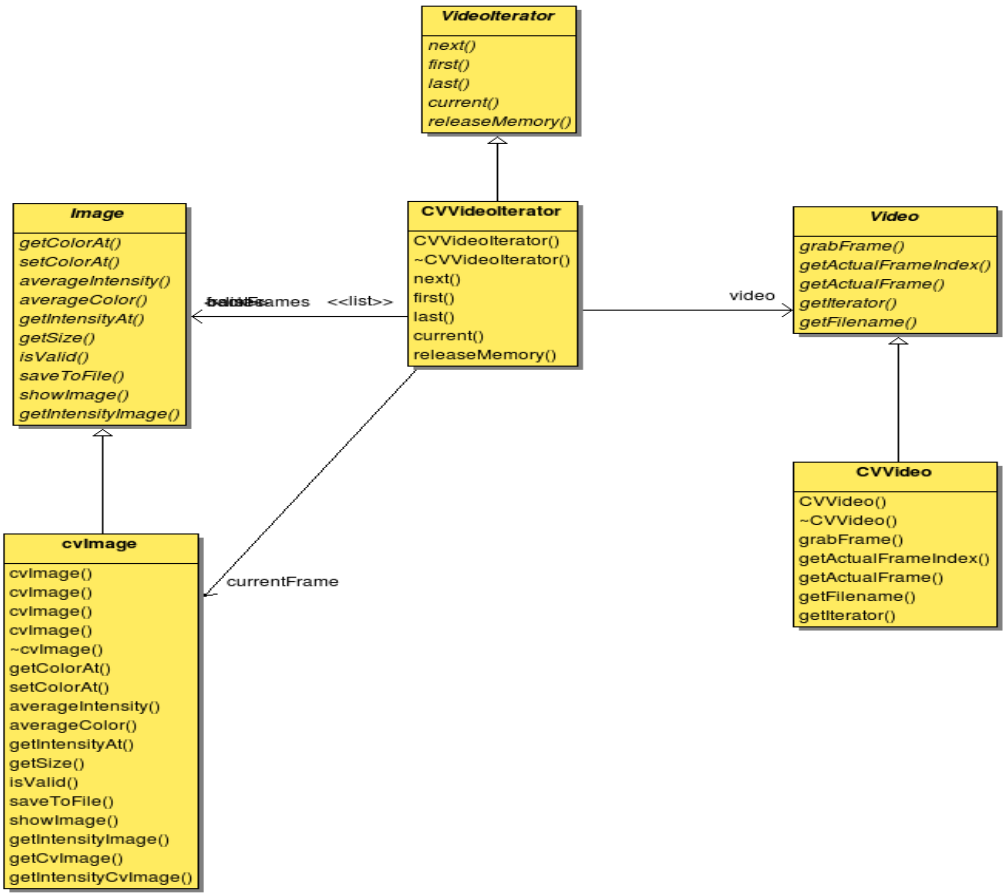

Figura 4.2: Diagrama de clases concretas para *Image* y *Video*.

4.4.2. Clase: *FrameExtractor*

Permite iterar sobre un directorio que contiene un conjunto de *frames* en forma de imágenes, con alguna extensión y prefijo particular. A la clase se le debe informar el tipo de extracción a realizar, para esto se ha usado el patrón de diseño *Command* [7], ver Figura 4.3.

En este caso se ha implementado la extracción *frame-by-frame*; se crea una instancia de la clase *FileIterator* y por cada archivo dentro del directorio que tenga el prefijo y la extensión indicada se crea una instancia de *Image*.

Esta clase fue diseñada para ser usada por las clases que implementan *FeatureExtractionMethod*, obteniendo los *frames* de un directorio, convirtiéndolos a objetos del tipo *Image* y luego operando sobre ellos mediante las funciones provistas por esta abstracción.

Los métodos que contiene son: *getNextDiskFrame()*, *framesNumber()*, *getUrlBase()* y *getActualFilename()*.

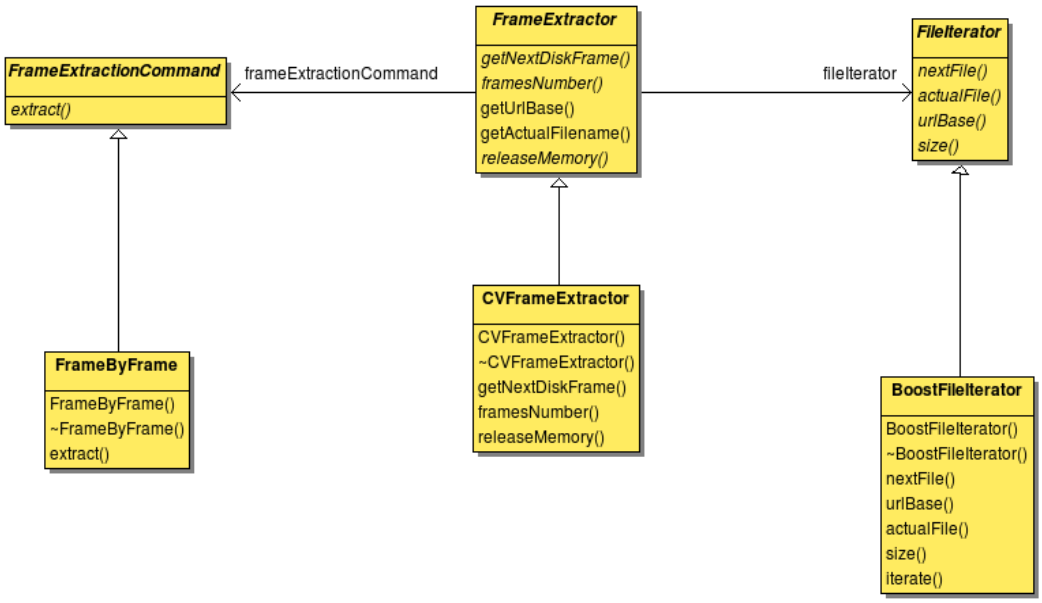

Figura 4.3: Diagrama de clases concretas para la *FrameExtractor*.

4.4.3. Clase: *FeatureExtractionMethod*

Permite extraer un vector característico a partir de un *frame*. Para ello se vale de una instancia de *Image* y la extracción desde los directorios de una instancia de *FrameExtractor*. Es una abstracción para toda clase que extraiga características. Supone dos métodos que deben ser implementados: *bool apply()*, que extrae las características de un directorio que contiene archivos de imágenes y se encarga de crear un archivo en disco, conteniendo los vectores característicos mediante una instancia de *Descriptor*; y *bool show()*, que se propone como posible método a implementar y cuya función es representar el descriptor en algún dispositivo de salida. En este caso se utilizó el patrón diseño *Factory Method* [7].

Las clases concretas que implementan *FeatureExtractionMethod* son (ver Figura 4.4):

EdgeFeature:

Recibe como parámetros una instancia de *FrameExtractor*, la cantidad de bloques en que se dividirá cada *frame* y el nivel de umbral T (*Threshold*). Obtiene uno a uno los *frames* desde el disco (en una instancia de *Image*), divide en bloques, obtiene las intensidades por regiones, finalmente aplica los *kernels* y extrae el mayor valor. Cada *frame* del que extrae características es guardado en disco mediante una instancia de *Descriptor*.

MeanAverageFeature:

Recibe como parámetros una instancia de *FrameExtractor* y la cantidad de bloques para dividir la imagen. Obtiene uno a uno los *frames* en una instancia de *Image*, divide en bloques, obtiene las intensidades por bloques usando los métodos de una instancia de *Image* y las va ingresando en una lista ordenada (para obtener el *ranking*). Cada descriptor de un *frame* es guardado en disco mediante una instancia de *Descriptor*.

BCSFeature:

Recibe como parámetros una instancia de *FrameExtractor*, una instancia del tipo *Descriptor* con un descriptor válido apuntando a un archivo en disco y una instancia del tipo *PCA* (*Principal Component Analysis*) para realizar el análisis de componentes principales (*PCA*). Este descriptor necesita una instancia de *Descriptor* de donde leer los datos del descriptor que se usará para hacer el BCS. La lectura se hace mediante la clase abstracta *FileReader*

implementada mediante *BinaryFileReader* (para lectura de datos binarios en disco) y pasada como parámetro dentro de una instancia de *Descriptor* del tipo *BCSDescriptor*. Para el BCS, esta clase recibe como parámetro un objeto del tipo *PCA*. Estos objetos son una abstracción para efectuar dicho análisis. En particular se implementa esta clase abstracta mediante la clase *cvPCA*, que hace uso de las funciones y métodos de la librería *OpenCV* para efectuar *PCA*. Cada *frame* del que se extrajeron sus características es guardado en un archivo binario mediante una instancia de *Descriptor*.

DCTfeature:

Este extractor de características recibe como parámetros: una instancia de *FrameExtractor*, la cantidad de coeficientes de la DCT a extraer y un objeto del tipo *DCT* con el cual se realizarán los cálculos. En este caso se ha desacoplado de la clase el cálculo de la DCT, de modo que posteriormente pueda ser reemplazado por otra implementación. La clase DCT es una abstracción que debe proveer las funciones para efectuar una transformada coseno discreta sobre una instancia de *Image*. En este caso, la clase concreta que implementa *DCT* es *cvDCT*, que hace uso de las funciones y métodos provistos por *OpenCV* para efectuar DCT. Al igual que las demás clases, guarda en disco los vectores característicos mediante una instancia de *Descriptor*.

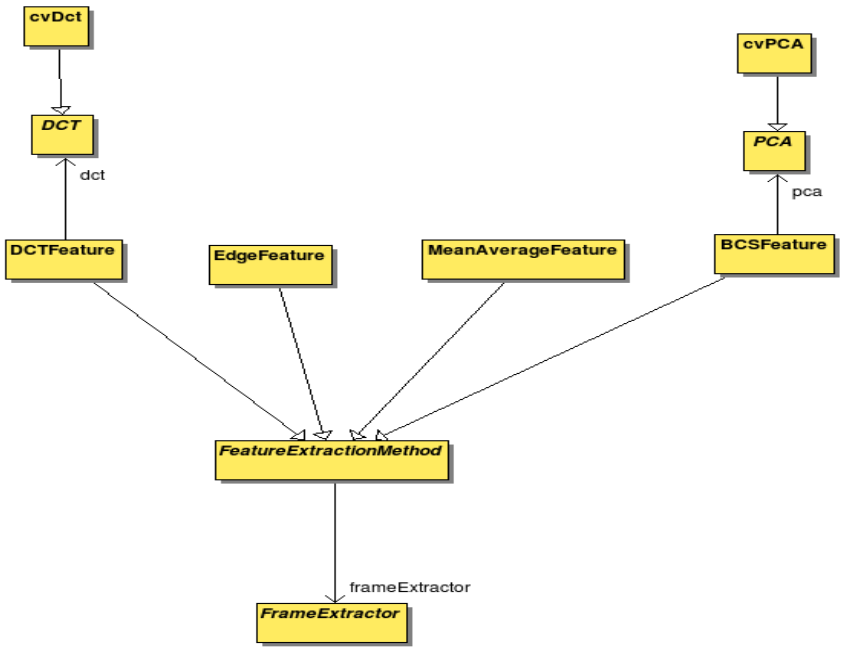

Figura 4.4: Diagrama de clases concretas para *FeatureExtractionMethod*.

4.4.4. Clase: *Descriptor*

La clase *Descriptor* (ver Figura 4.5) es una clase que declara un conjunto de métodos y los implementa pero permite la re-escritura de los mismos. Si una clase desea usar los métodos provistos por esta clase, simplemente deriva de ella y no implementa los métodos. En caso contrario, deriva de ella e implementa los métodos a modo de reemplazar los anteriores.

La clase *Descriptor* provee un conjunto de métodos para apoyar la lectura y escritura a disco desde una instancia de *FeatureExtractionMethod*. La implementación se basa en la instanciación de la clase *FileReader* y *FileWriter* mediante *BinaryFileReader* y *BinaryFileWriter*. Su intención es proveer una estructura simple para guardar y leer los vectores característicos en un archivo binario. El archivo que escribe contiene una cabecera, mediante un entero correspondiente a la dimensión de los vectores a guardar y uno para el número de vectores característicos a guardar -en general uno por cada *frame*-. Seguido a la cabecera, se guardan en forma continua los vectores característicos utilizando el tipo *double*.

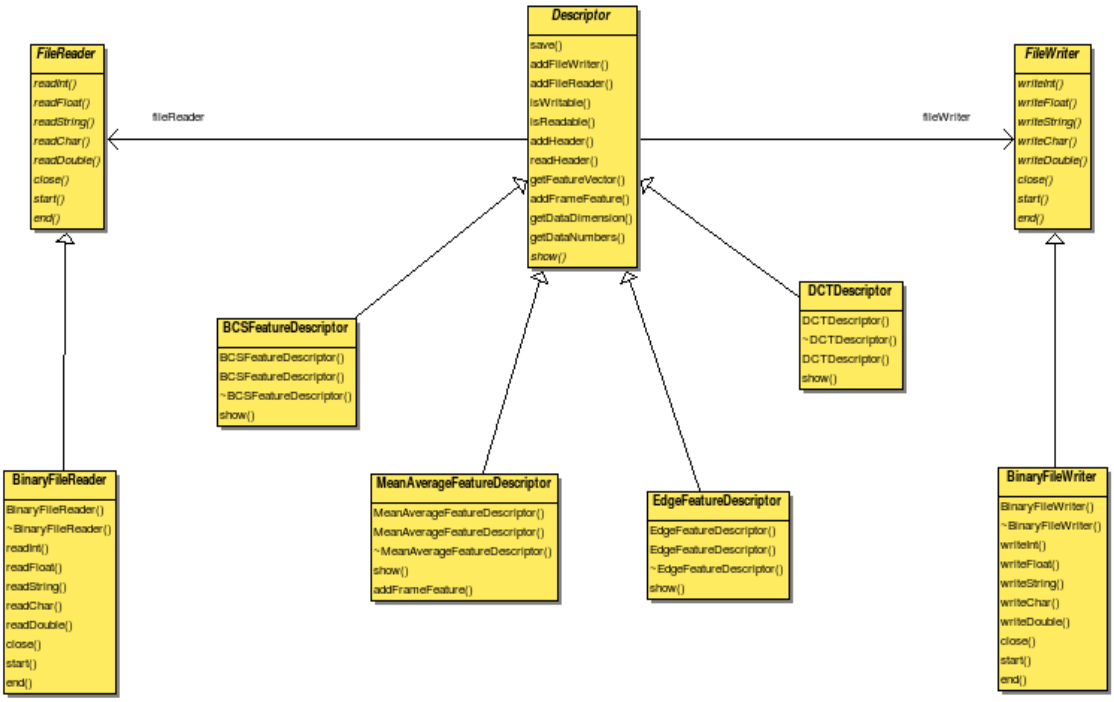

Figura 4.5: Diagrama de clases concretas para la *Descriptor*.

4.4.5. Clase: *Distances*

Esta clase es puramente abstracta y sólo se utiliza para declarar una interfaz de cómo obtener el resultado de una comparación de dos descriptores.

La clase tiene un único método a implementar, *double calculate(Descriptor *descriptor)*, que recibe un *Descriptor* y devuelve un *double* que representa la distancia calculada. La instancia *Descriptor* se asocia al vector característico (o matriz característica) del objeto por el que se está buscando y se debe pasar el objeto a comparar implementando un método del tipo *set*. De esta forma, la jerarquía de clases sugiere que se cree una vez una instancia de la clase *Distances* que contenga el objeto de consulta y en usos sucesivos del método *calculate* se obtengan las distancias a un conjunto de objetos de comparación.

El diagrama para esta clase junto con las clases concretas que la implementan se muestra en la Figura 4.6.

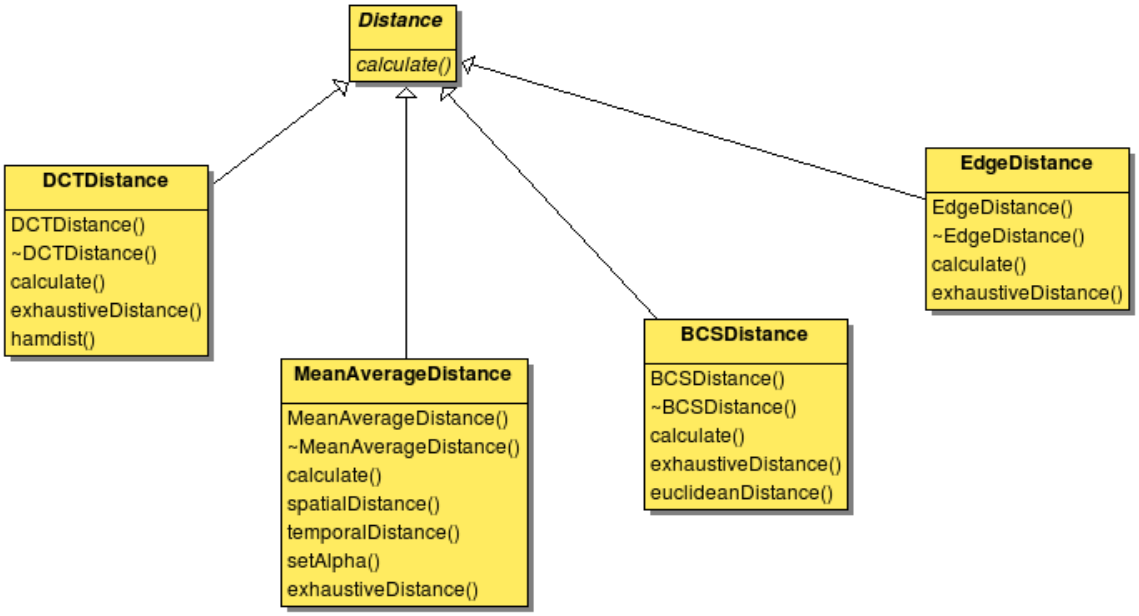

Figura 4.6: Diagrama de clases concretas para *Distances*.

4.4.6. Ejecución

Todas las clases se agrupan en una librería estática: *VideoDescriptor*. Dentro de esta se encuentran todas las clases mencionadas en la Sección 4.4 del Capítulo 4 más las clases que soportan a estas (tales como *DCT*, *PCA*, *FileReader*, *FileWriter* entre otras). Para utilizar

cualquiera de estas clases se debe incorporar al código esta librería estática. En particular, se implementaron programas -uno por cada descriptor- que iteran sobre un directorio base que contiene imágenes -correspondientes a *frames* de un video- calculan las características correspondientes y lo guardan en un archivo binario.

4.4.7. Detalles de implementación

Para *Edge* los autores hablan de un parámetro T (umbral) que determina si un bloque es clasificado en uno u otro gradiente de intensidad. Sin embargo no especifican qué valor debe tener T. Para obtener una aproximación de un valor adecuado de T se ejecutó el cálculo del descriptor sucesivamente y se observaron los valores arrojados. Para valores $T \geq 100$ el descriptor no logra clasificar ni el 90 % de los bloques. Para valores $10 \leq T \leq 100$ el descriptor clasifica cerca del 80 % de los bloques por cada *frame*. Otra decisión "arbitraria" se tomó en la función de similitud utilizada en este descriptor; en ésta los autores no mencionan cómo se trata el caso en que $x_i = 11$ y/o $y_i = 11$. En este caso se decidió hacer que si $x_i = 11$ y/o $y_i = 11$ entonces $E_i = 0$, puesto que dos bloques no clasificados, no necesariamente son similares.

Capítulo 5

Evaluación experimental

5.1. Objetivos experimentales

El objetivo general de la evaluación experimental es la comparación de los descriptores en cuanto a eficiencia y eficacia. Además se debe verificar que cada descriptor sea correcto y robusto.

Para cada medición (eficiencia, eficacia, consistencia y robustez) se necesitan valores cuantitativos. Sin embargo, es difícil evaluar cuantitativamente consistencia y robustez, pues ambas características se basan en lo que cada autor definió en sus investigaciones. Es por ello que se plantea una medición basada en las siguientes definiciones:

Correctitud: Evaluación binaria. Cumple o no cumple con las metas propuestas por el autor. El objetivo es meramente verificar que la implementación del descriptor es correcta respecto a lo propuesto por el autor del mismo, mediante pruebas sobre videos en que los vectores característicos se pueden predecir de antemano. De esta forma, se crean videos considerando el análisis que hará el descriptor del mismo, y si la implementación está correcta entonces el vector característico coincidirá con lo esperado.

Eficiencia: *Ranking* de uso de recursos respecto a: CPU (a nivel de usuario y a nivel de sistema o *kernel*), uso de disco (lectura y escritura) y capacidad de procesamiento (*frames* procesados por segundo y *Mb* procesados por segundo). Los mejores rankeados (más cercanos al primero) se considerarán más eficientes.

Eficacia y Robustez: *Ranking* de las distancias de un video original a un conjunto de videos (entre ellos el original con transformaciones). Mientras más cercano al primero

esté un video transformado respecto de su original, más robusto se dirá el descriptor. Para cuantificar eficacia, se realizarán gráficos de *Precison vs Recall* utilizando promedios sobre todos los resultados en cada descriptor.

5.2. Planteamiento de los experimentos

5.2.1. Correctitud

1. ***Edge descriptor (Figura 5.1):*** El objetivo es saber si los *kernels* están detectando los bordes de cada *frame*. Cabe destacar que los filtros de borde de este descriptor están pensados para detectar cambios de intensidades de los sub-bloques, tal como se muestra en la Figura 5.1. Para ello se disponen varias imágenes con patrones de intensidades con distinta orientación. En este caso, se espera obtener en cada uno de los cuadrantes de cada imagen, el identificador correspondiente al *kernel* con la misma orientación que la figura dibujada sobre el mismo. Notar que el descriptor pretende calificar a un cuadrante en uno de los 11 puestos si y sólo si el gradiente de ese cuadrante (dirección de aumento en la intensidad de luminosidad) coincide con alguno de los filtros aplicados (la coincidencia se hace mediante ponderación de cada uno de los 4 sub-bloques y la suma correspondiente).

2. ***Mean average descriptor (Figura 5.2):*** Dado que este descriptor se basa en las intensidades de cada cuadrante, se dispone un conjunto de imágenes que simulan un video en el que un círculo blanco sobre fondo negro, se desplaza por la pantalla, usando un cuadrante por cada frame. Se espera obtener en cada descriptor de cada *frame*, el índice 1 en la misma posición en que se encuentra el círculo blanco.

3. ***BCS descriptor (Figura 5.3):*** Tal como se describió en la Sección 3.3, este descriptor hace un *PCA* sobre los vectores característicos y guarda como nuevo vector característico los componentes principales. De esta forma nos interesa saber si el análisis de componentes principales se está haciendo de forma correcta. Se da entonces como entrada al descriptor un conjunto de puntos en 2 dimensiones - que simulan un descriptor 2-dimensional - todos con las mismas coordenadas. Deben obtenerse vectores con todas las componentes cero para cada dimensión. Además se dará como entrada el

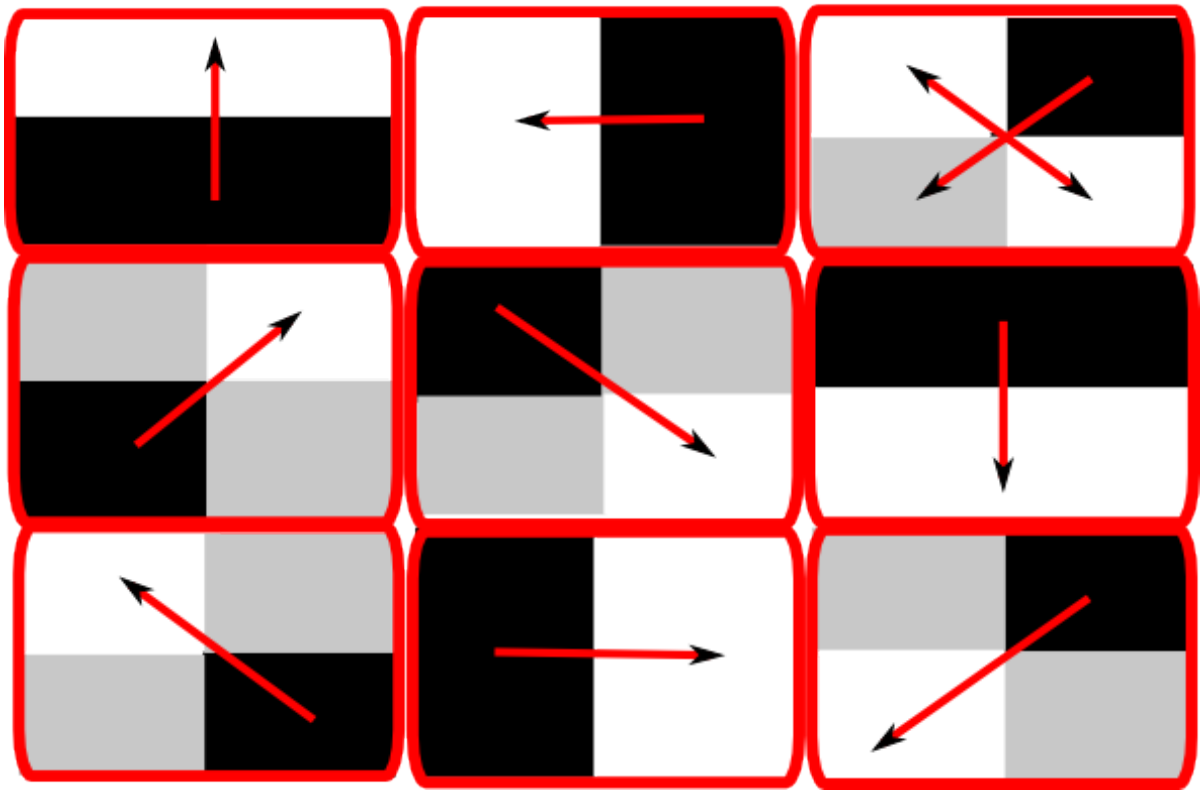

Figura 5.1: En la figura para el experimento de correctitud de *Edge descriptor* se han dispuesto 3×3 cuadrados con distintos gradientes de intensidades (de izquierda a derecha, arriba hacia abajo): $90°$, $180°$, no direccional, $45°$, $-45°$ $-90°$, $135°$, $0°$, $-135°$. Además se ha destacado en la figura la dirección del gradiente de intensidad mediante flechas (éstas no se consideran al calcular el descriptor y están sólo para clarificar el concepto). El experimento debiera arrojar en cada uno de los cuadrantes el valor asociado al ángulo de la orientación que se presenta en él, o en el peor de los casos uno muy cercano. Los valores para el experimento en este caso serán de $N = 3$ y el umbral será de $T = 250$, puesto que el constraste en la imagen es alto.

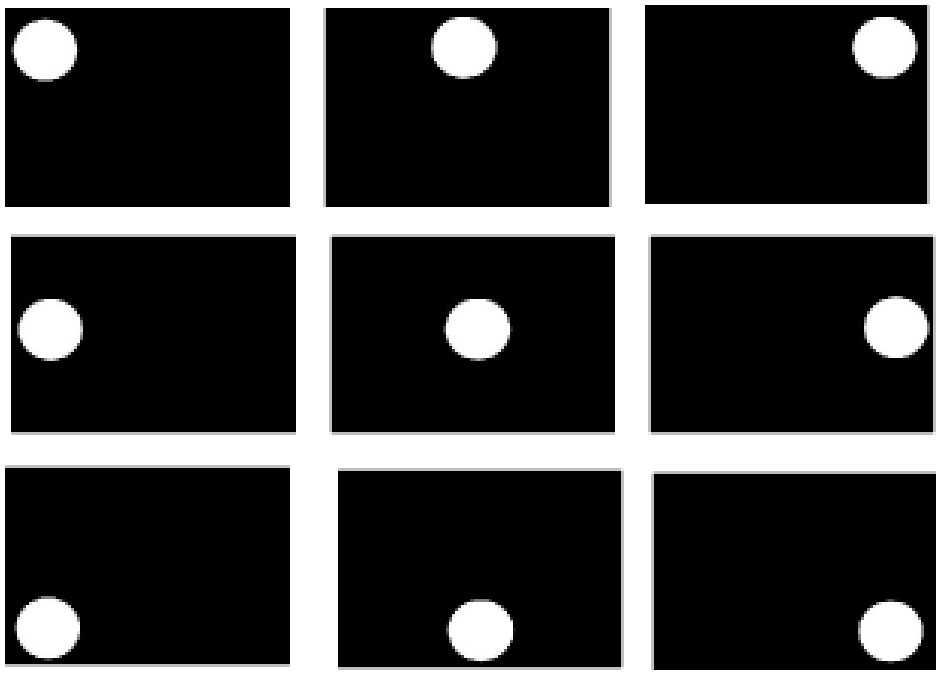

Figura 5.2: En la figura para el experimento de *Mean average descriptor* se ha creado un video con 9 *frames*. En cada uno se ubica un círculo con intensidad máxima ($I := 255$) que varía de posición entre cada *frame*. El experimento de correctitud, debería arrojar en la posición uno el bloque donde se ubica el círculo, considerando una partición de los *frames* de 3×3.

caso en que todos los descriptores 2-dimensionales forman en conjunto un círculo sobre el plano en $\mathbb{R}^2$. En este caso deben obtenerse vectores en las mismas direcciones que los ejes coordenados del sistema. Finalmente se aplica el descriptor sobre un conjunto de descriptores 2-dimensionales, ubicados sobre una recta con pendiente sobre el plano en $\mathbb{R}^2$; en este caso se espera obtener como respuesta un eje en la dirección de la recta y otro perpendicular a este.

4. ***DCT descriptor: (Figura 5.4)*** Dado que *Discrete Cosine Transform* (DCT) es una transformación que cuenta con inversa sin pérdida de información, se puede recuperar la imagen original usando todos los coeficientes de la DCT. Se usará entonces la aplicación de la inversa DCT sobre un conjunto de imágenes. Para ello, se aplicará el descriptor sobre un conjunto de *frames*, usando como parámetro de entrada todos los coeficientes y luego se aplicará la inversa sobre éstos. Notar que la imagen restaurada no es la misma que la original, debido a que el descriptor aplica una fuerte discretización sobre los coeficientes obtenidos de la DCT. Sin embargo, se debiese obtener una imagen en escala de grises (en la discretización se elimina la información de colores) en donde se

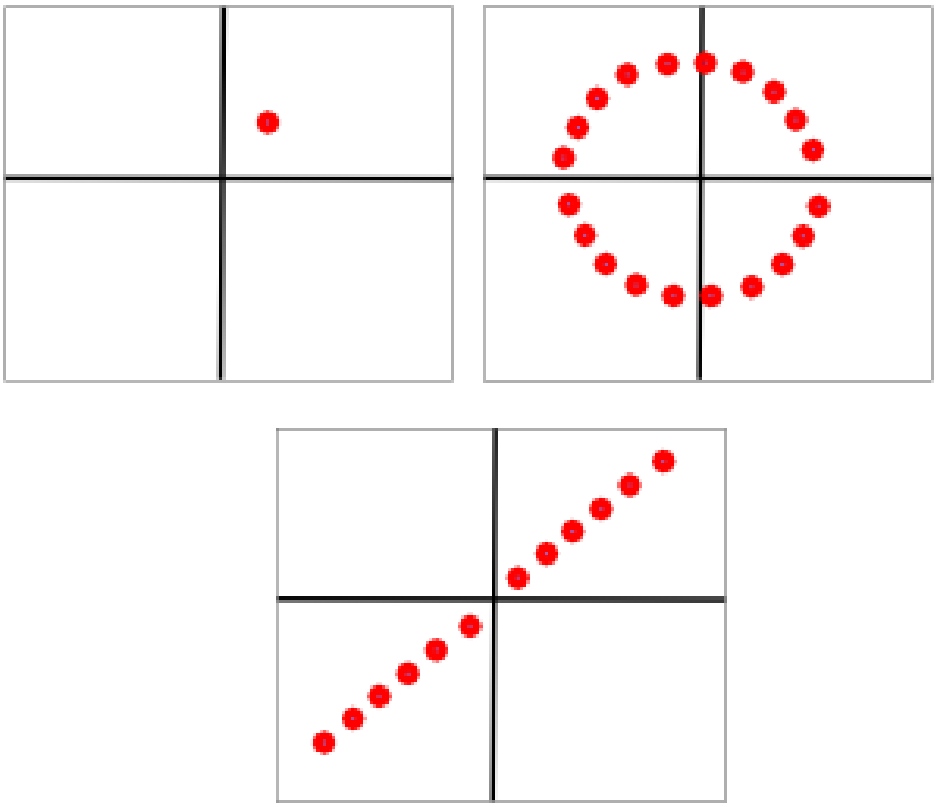

Figura 5.3: En la figura para el experimento de correctitud de *BCS descriptor*, se ubican puntos en el espacio bi-dimensional. Cada punto representa un vector de características para un *frame* particular de un video. Se espera que la obtención de características, arroje un descriptor tal que para el primer caso (arriba, izquierda) se obtenga un descriptor cero (pues no hay varianza entre los *frames*). Para el segundo caso (derecha, arriba), un descriptor con los mismos ejes coordenados del sistema. Para el último caso (abajo), un descriptor con su eje principal sobre la línea recta definida por los puntos.

Disco	**CPU**
Lecturas (Kb/s)	Nivel usuario (% de tiempo)
Escrituras (Kb/s)	Nivel de sistema (% de tiempo)
Espacio en disco (Mb)	Nivel de máquina virtual (% de tiempo)
	Total CPU/procesador (% de tiempo)

Cuadro 5.1: Principales mediciones que se utilizaron para medir eficiencia.

mantiene la información relativa a los bordes más significativos de la imagen original.

Figura 5.4: Imagen utilizada para verificar correctitud en DCT descriptor.

Los resultados obtenidos junto a un breve análisis se detallan en la Sección 5.4.1.

5.2.2. Eficiencia

Se aplican las funciones de extracción de características sobre un conjunto de videos. Por cada extracción se mide el uso de recursos mediante el comando linux `time`. Este comando entrega las estadísticas de disco, memoria y cpu. El proceso se repite por todos y cada uno de los videos, para contar con un promedio y varianza del consumo de recursos. En el Cuadro 5.1 se muestran los valores medidos para cada uno de los recursos.

Los resultados obtenidos junto a un breve análisis se detallan en la Sección 5.4.2.

5.2.3. Eficacia y Robustez

A un video particular se le aplica un conjunto de transformaciones. Para cada uno de ellos (incluidos los videos originales sin transformaciones) se calculan los descriptores. Para cada video transformado, se calcula la distancia respecto a todos los demás videos (transformados y originales), se ordenan las distancias y se crea un *ranking* de estas por cada tupla (video, transformación, descriptor). Las transformaciones deben estar situadas en el contexto de los videos digitales y ser comunes de encontrar en programas de televisión, Internet y cine por ejemplo. Para ello se han escogido doce transformaciones (ver Capítulo 2.2) aparecidas en Trecvid 2008 [1].

Para comparar robustez se escoge un video $V_q \in V$, una transformación $t_q \in T$ y un descriptor $d_q \in D$ (siendo V, T y D las colecciones de videos, transformaciones y descriptores). Se buscan las posiciones en el *ranking* de los videos v_s tal que $v_s = v_q$ con transformaciones t_s tal que $t_s \neq t_q$ y descriptor d_s tal que $d_s = d_q$. Se repite esta operación para todo $v_q \in V$ y para todo $t_q \in T$ para obtener los *ranking* de todos los videos relevantes (mismo video que el de consulta pero con distinta transformación) para cada uno de los videos transformados de la base de datos de un descriptor dado. Finalmente se promedia el *ranking* de cada una de las transformaciones de todos los videos. Esto se repite por cada descriptor.

Para comparar eficacia se realiza una medición de *Precision-Recall* utilizando todos los resultados de correctitud.

Los resultados obtenidos se presentan en la Sección 5.4.3.

5.3. Marco de trabajo

5.3.1. Biblioteca de videos

Se escogió como base de datos de videos la biblioteca *Muscle Video Copy Detection 2007*. La biblioteca se pide de manera personal por *email* a través de la página web [2]. *Muscle* fue diseñada especialmente para la conferencia internacional ACM de recuperación de video *CIVR 2007*.

Esta librería consta de 101 videos con tamaños desde los 5 MB hasta algunos GB. Los

[1] http://www-nlpir.nist.gov/projects/tvpubs/tv8.slides/CBCD.slides.pdf
[2] http://www-roc.inria.fr/imedia/civr-bench/data.html

videos provienen de distintas categorías, las que van desde los documentales hasta las películas animadas. Todos los videos pueden ser usados sólo para investigación y citando previamente la proveniencia de los mismos.

De estos videos se escogieron los 30 más pequeños, que forman una subcolección de aproximadamente 1.5 Gb y una duración total de un poco más de 2 horas, equivalentes a alrededor de 200.000 *frames*. Cada *frame* tiene dimensiones de 352×288 con un peso de 30 KB aproximadamente (excluyendo al audio). A cada uno de estos videos se le aplicaron las 12 transformaciones de la Figura 2.2 página 8. Con esto se tiene una base de datos de 360 videos, de los cuales 30 son los videos originales de *Muscle* y 330 corresponden a los videos transformados. Por consiguiente, al aplicar los descriptores se obtienen $12 \times 30 \times 4$ archivos que describen cada uno de los 360 videos. En total estos videos suman alrededor de 2 millones de *frames* utilizando un total aproximado de 40 GB en disco.

5.3.2. Hardware

Para el pre-procesamiento de videos (aplicación de transformaciones a los videos) y el cálculo de las distancias entre los descriptores, se utilizó el servidor *Wedge* del grupo Prisma[3] del Departamento de Ciencias de la Computación - Universidad de Chile - el cual está dedicado entre otras tareas al análisis de imágenes y video. El servidor consta de un procesador Intel de 64 bits del tipo QuadCore con sistema operativo CentOs. Sin embargo, el código para la extracción de características no se pudo compilar en esta máquina, puesto que las versiones de las dependencias de OpenCV no coincidían con las que trae por defecto CentOs. Luego, para el cálculo de los descriptores se utilizó un desktop Intel Core 2 Duo de 32 bits de $1,8$ Ghz y 2 Gb de memoria RAM.

5.3.3. Ejecución de los experimentos

El primer paso fue obtener los *frames* desde los videos; para esto se usó *ffmpeg* [4]. Se crearon scripts para obtener las transformaciones de los videos a partir de sus *frames* utilizando *ffmpeg* y el comando *Convert* de la librería *ImageMagick* de Linux. Todos ellos se comprimieron en archivos *tar* separados (uno por cada video) y se guardaron en un disco duro externo.

[3] http://prisma.dcc.uchile.cl
[4] http://ffmpeg.org/

Para el cálculo de descriptores, se creó un *script bash* que por cada archivo comprimido descomprime al disco duro interno y ejecuta el cálculo de cada uno de los descriptores sobre ellos; esto pues los archivos de los *frames* usaban gran cantidad de espacio en disco y eran muchos archivos, lo que deficultaba la navegación sobre esos directorios. Cada ejecución del cálculo de un descriptor es un comando independiente monitoreado por el comando *time* de Linux. La salida de este comando es redirigida a un archivo del tipo *csv* (*Comma separated value;* valores separados por coma). Finalmente se eliminan los *frames* y se continúa con el siguiente video. El computador no tenía más tareas en proceso.

Para el cálculo de distancias, se creó un *script bash* que por cada video calcula la distancia de los descriptores a cada uno de los demás videos. La distancia se guarda en un archivo *csv* junto con los nombres de los dos videos comparados. El servidor *wedge* mantenía otras tareas de otros usuarios mientras se ejecutaba este proceso, pero esto no afecta las mediciones del tiempo de los cálculos de distancias puesto que dicho cálculo no fue monitoreado con *time*.

Después de la ejecución se obtuvieron 360 archivos *csv* para cada uno de los 4 descriptores; esto es 1440 archivos, cada uno con 360 líneas que indican la distancia entre un video y otro. Para resumir toda esta información, se creó una base de datos relacional (usando *MySql* [5]), usando el esquema de la Figura 6.2 del Apéndice. Con esta base de datos se puede resumir fácilmente toda la información para un descriptor, mediante una consulta en lenguaje *sql*.

Para completar las tablas de la base de datos, se insertaron los nombres de los videos (en *movies*), las 12 transformaciones (más el tipo *"NONE"* para indicar los videos originales) y los descriptores. Los resultados de distancias se insertaron mediante un programa en *Java* que abre cada uno de los archivos *csv* e inserta los datos en la tabla *distancesString*. Finalmente se creó la tabla *rankingDistances* mediante una consulta *sql* (ver el Apéndice B .1).

Todas las consultas usadas para obtener los datos resumidos desde la base de datos se adjuntan en el Anexo B .1.

[5]http://dev.mysql.com/

5.4. Resultados

5.4.1. Correctitud

Edge descriptor

El vector característico V_{edge} de la Ecuación 5.1, es el resultado del experimento propuesto en la Sección 5.2.

$$V_{edge} = \begin{bmatrix} 3 & 5 & 9 & 2 & 8 & 7 & 4 & 1 & 6 \end{bmatrix} \tag{5.1}$$

V_{edge} indica la clasificación de cada bloque (orientaciones del gradiente de intensidades), y $G(V_{edge})$ en la Ecuación 5.2 muestra las direcciones del gradiente que corresponden a las obtenidas en V_{edge}.

$$G(V_{edge}) = \begin{bmatrix} 90° & 180° & ND1 & 45° & -45° & -90° & 135° & 0° & -135° \end{bmatrix} \tag{5.2}$$

Al aumentar el valor a $T = 600$ -para verificar la correctitud del parámetro T de umbral- el vector V'_{edge} de la Ecuación 5.3 no obtiene ningún calce para los cuadrantes. Esto se debe a que ningún cuadrante alcanza a tener un valor para el filtro mayor al T propuesto.

$$V'_{edge} = \begin{bmatrix} 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 \end{bmatrix} \tag{5.3}$$

Al comparar la Figura 5.1 con el vector V_{edge} y $G(V_{edge})$ obtenido de las Ecuaciones 5.1 y 5.2, se aprecia el primer bloque de la figura (arriba izquierda) detectado como perteneciente a los 90°, esto se debe a que existe un gradiente de intensidades con esa dirección (del negro al blanco). Para el segundo bloque, existe un gradiente de intensidades en la dirección 180°, en el tercer bloque no se puede determinar ningún gradiente predominante, en el cuarto bloque la gradiente tiene dirección de 45°, en el quinto bloque $-45°$, en el sexto el gradiente está en $-90°$, en el séptimo 135°, el octavo 0° y en el noveno $-135°$. Esto concuerda con los gradientes de intesidad de la Figura 5.1.

Al aumentar el valor T se obtienen sólo valores 11 (Ecuación 5.3) lo cual tiene sentido, pues para este alto incremento del umbral no hay bloque de la imagen que convolucionado con algún filtro alcance a superar dicho umbral, y por ende ningún cuadrante es candidato para ningún filtro.

De este modo el descriptor se considera correcto en cuanto al parámetro umbral T y su capacidad de clasificar bloques de la imagen en cuanto a los gradientes de intensidades en ellas.

Mean average descriptor

Usando como entrada un video con los *frames* de la Figura 5.2, se obtiene la matriz M_{mean} donde cada fila i-ésima representa el descriptor del *frame* i-ésimo en la Ecuación 5.4.

Observando la diagonal de la matriz M_{mean} de la Ecuación 5.4 se ve (destacado en ***cursiva***) cómo cambia el descriptor cada vez que el círculo blanco cambia de posición. En todos los *frames* (filas del descriptor), se obtiene una intensidad máxima (valor 1) que cambia de posición entre *frame* y *frame*; cada una de estas posiciones corresponde al cuadrante en donde se ubica el círculo de alta intensidad (Figura 5.2).

$$M_{mean} = \begin{bmatrix} \boldsymbol{1} & 8 & 5 & 2 & 7 & 4 & 9 & 6 & 3 \\ 9 & \boldsymbol{1} & 4 & 8 & 5 & 3 & 7 & 6 & 2 \\ 9 & 6 & \boldsymbol{1} & 8 & 5 & 2 & 7 & 4 & 3 \\ 8 & 7 & 4 & \boldsymbol{1} & 6 & 3 & 9 & 5 & 2 \\ 9 & 6 & 4 & 8 & \boldsymbol{1} & 3 & 7 & 5 & 2 \\ 9 & 6 & 3 & 8 & 5 & \boldsymbol{1} & 7 & 4 & 2 \\ 9 & 7 & 4 & 8 & 6 & 3 & \boldsymbol{1} & 5 & 2 \\ 9 & 6 & 4 & 8 & 5 & 3 & 7 & \boldsymbol{1} & 2 \\ 9 & 6 & 3 & 8 & 5 & 2 & 7 & 4 & \boldsymbol{1} \end{bmatrix} \tag{5.4}$$

El descriptor está entonces clasificando correctamente los bloques de una imagen de acuerdo a sus intensidades y asignándoles posiciones relativas en el vector característico.

BCS descriptor

Para el conjunto de datos que forman una circunferencia (Figura 5.3) se obtiene el descriptor M_{BCS} de la Ecuación 5.5. La proyección del mismo en el plano cartesiano se presenta

en la Figura 5.5. La mayor varianza se da en dos direcciones ortogonales (puesto que el descriptor es una circunferencia); esto queda representado en un par de ejes que coinciden con los ejes cartesianos del plano.

$$M_{BCS_1} = \begin{bmatrix} -9.09161e-16 & -70.7107 \\ -70.7107 & 9.09161e-16 \end{bmatrix} \tag{5.5}$$

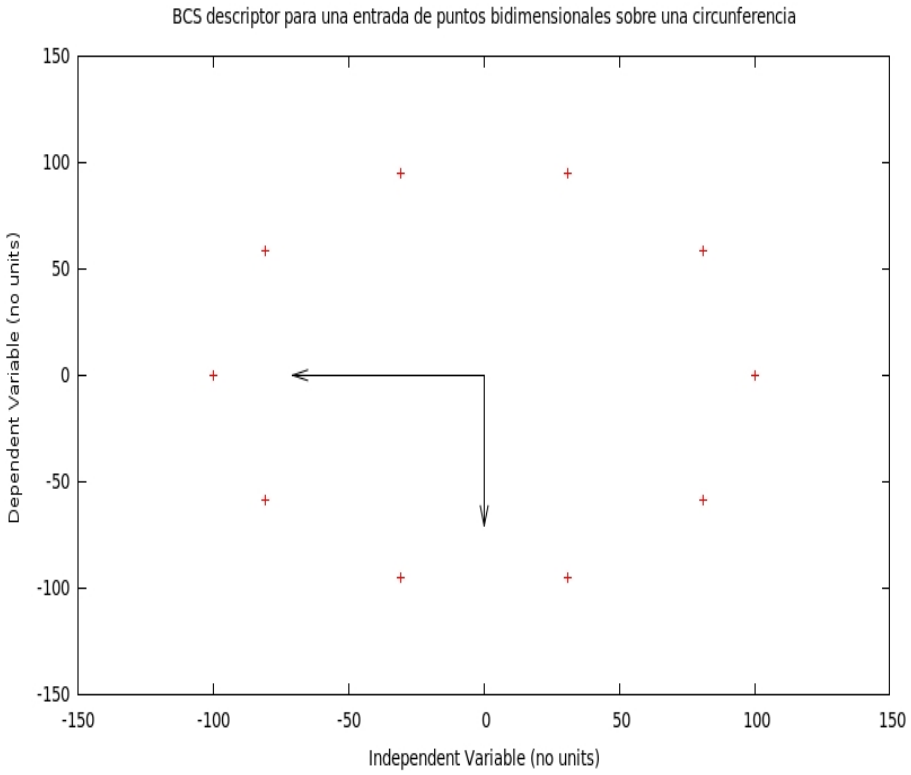

Figura 5.5: Gráfico de los vectores resultantes del *BCS descriptor* aplicado sobre un descriptor ficticio bidimensional. Los datos originales están sobre una circunferencia y se muestran con signo '+' en la gráfica.

Para el caso en que los datos forman una línea recta, se obtiene el vector característico de la Ecuación 5.6 que se ha proyectado en el plano cartesiano en la Figura 5.6. La mayor varianza está en la dirección de la línea recta (primer vector) y el segundo vector tiene tamaño muy cercano a cero, dado que en esta dirección -perpendicular al primer vector- no existe varianza alguna en los datos.

$$M_{BCS_2} = \begin{bmatrix} -2.87228 & -287.228 \\ -2.67453e-14 & 2.67453e-16 \end{bmatrix} \tag{5.6}$$

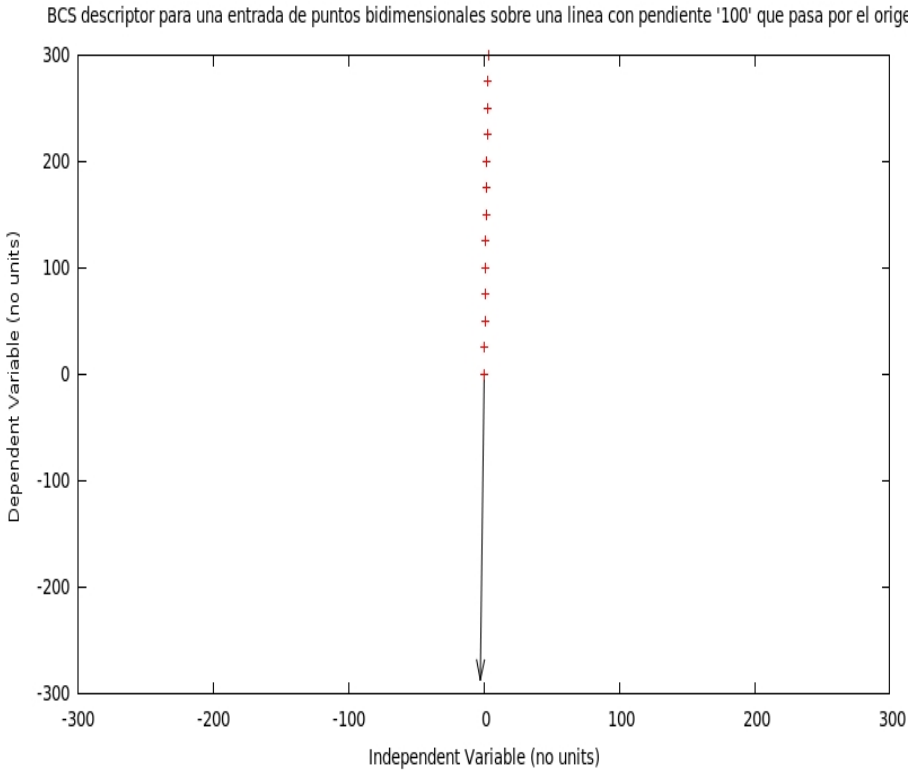

Figura 5.6: Gráfico de los vectores resultantes del *BCS descriptor* aplicado sobre un descriptor ficticio bidimensional. Los datos originales están sobre una línea recta con pendiente 100 que pasa por el origen. Los datos de la recta se muestran con signo '+' en la gráfica.

Finalmente cuando todos los datos están en un mismo punto, se obtiene el origen del sistema (vector característico M_{BCS_3} en la Figura 5.7). Los ejes tienen longitud cero, centrados en el origen, puesto que no existe varianza en los datos.

$$M_{BCS_3} \begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix} \tag{5.7}$$

De esta forma la implementación del descriptor calcula correctamente el BCS basado en *PCA*.

DCT descriptor

Al aplicar este descriptor sobre la imagen mostrada en la Figura 5.4, se obtienen las matrices características de las Ecuaciones 5.8, 5.9 , 5.10 y 5.11 correspondientes al cálculo con coeficientes 25, 36, 64 y 100 respectivamente.

Con estos vectores se calcula la antitransformada de la transformada coseno discreta ($IDCT$) para recuperar la imagen original (Figura 5.7). En esta imagen obtenida existe pérdida de información debido a dos factores: primero pues la DCT se aplica sobre el canal de intensidades y segundo pues se aplica una fuerte cuantización sobre los coeficientes originales de la DCT (Ecuación 3.12).

$$M_{DCT_1} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.8}$$

$$M_{DCT_2} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.9}$$

$$M_{DCT_3} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.10}$$

$$M_{DCT_4} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.11}$$

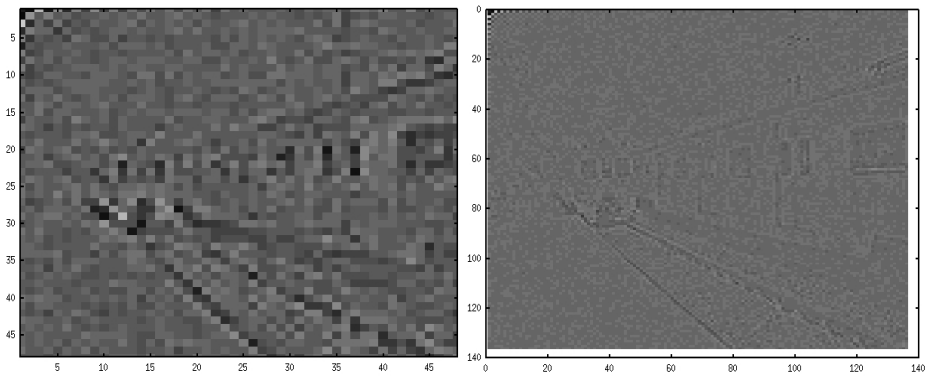

Figura 5.7: Resultado de aplicar *IDCT* sobre vectores resultantes de *DCT descriptor*. En este caso se han usado vectores de dimensión 1×1024 y 1×18496 para apreciar mejor la recuperación de la imagen original. Notar que la mejor imagen que se puede obtener es generando un vector de dimensión 1×409.600 (lo que equivale a un vector con tantas dimensiones como píxeles tenga la imagen original).

Descriptor	**Fps**	**Inf**	**Sup**
BCS	$78.828,0$	$78.656,0$	$78.999,0$
Mean Average	$58,6$	$56,5$	$60,7$
Edge	$58,4$	$56,3$	$60,5$
DCT-based	$25,8$	$25,0$	$26,6$

Cuadro 5.2: Promedios de la capacidad de procesamiento de cada descriptor (cantidad de *frames* por cada segundo) y sus intervalos de confianza (95 %).

5.4.2. Eficiencia

Los Cuadros 5.2 a 5.6 muestran los promedios de los experimentos de eficiencia junto con sus intervalos de confianza. Los promedios se obtuvieron como resultado de calcular el descriptor correspondiente, sobre los 360 videos (transformados más originales). En líneas horizontales se marca el promedio alcanzado por toda la muestra y en líneas verticales el intervalo de confianza al 95 %. El tamaño de cada video se obtuvo mediante el comando *avinfo* que resume información de videos (el audio está considerado dentro del tamaño, lo cual incide en los promedios, sin embargo es mucho menor al tamaño de los frames).

Del Cuadro 5.2 se puede obtener una comparación de la rapidez del cálculo de los descriptores. BCS es el más rápido (tanto en FPS como en Mb/s), superando por unas 10.000 veces la magnitud la velocidad de reproducción de un video (24 a 30 fps) y la de los demás descriptores. Luego siguen *Mean Average*, *Edge* y finalmente DCT. Debe notarse que para DCT se están calculando todos los coeficientes de la transformada coseno discreta aún cuan-

Descriptor	MB/s	Inf	Sup
BCS	564, 0	537, 0	591, 0
Mean Average	0, 44	0, 40	0, 48
Edge	0, 44	0, 40	0, 48
DCT-based	0, 19	0, 18	0, 19

Cuadro 5.3: Promedios de la capacidad de procesamiento (MB procesados por cada segundo) de cada descriptor y sus intervalos de confianza (95 %).

Descriptor	*CPU* (%)	Inf	Sup
BCS	99, 25 %	98, 74 %	99, 75 %
Mean Average	99, 08 %	99, 05 %	99, 11 %
DCT-based	99, 01 %	99, 00 %	99, 02 %
Edge	98, 92 %	98, 84 %	98, 99 %

Cuadro 5.4: Promedios del uso de *CPU* de cada descriptor y sus intervalos de confianza (95 %).

Descriptor	Escrituras en Disco / s	Inf	Sup
DCT-based	2, 438	1, 949	2, 926
Mean Average	0, 759	0, 665	0, 853
Edge	0, 211	0, 195	0, 227
BCS	0, 006	0, 005	0, 007

Cuadro 5.5: Promedios del uso de disco (escritura) de cada descriptor y sus intervalos de confianza (95 %).

Descriptor	Espacio total usado (KB)
DCT-based	1.057.672, 0
Mean Average	534.432, 0
Edge	150.316, 0
BCS	1.316, 0

Cuadro 5.6: Espacio total en disco (en KB) usado por los descriptores para toda la colección de videos.

do sólo se estén usando algunos de ellos. *Mean Average* supera a DCT y *Edge* en rapidez de cálculo debido a que *Edge* aplica sucesivamente filtros sobre los cuadrantes. Los filtros implican multiplicaciones y sumas, a diferencia de *Mean Average* que sólo utiliza promedios (sumas y una división por cada cuadrante).

En cuanto a la cantidad de escrituras en disco, BCS sigue siendo el más competitivo, puesto que sólo tiene que guardar tantos valores como dimensiones tenga el descriptor usado. En este caso, el descriptor usado es *edge*, es decir $n = 9$. DCT es el que más escribe en disco, esto pues por cada *frame* escribe 64 valores. *Edge* y *Mean* son similares en la cantidad de escrituras a disco, al tener que escribir por cada *frame* $n = 9$ y $n = 16$ respectivamente. Esto explica también el uso total en disco considerando toda la colección de videos.

En cuanto al uso de CPU, no hay grandes diferencias entre los descriptores, sí cabe destacar que por simplicidad en la escritura del código, no se usaron *threads* ni uso paralelo de los núcleos.

En cuanto a las complejidades de los descriptores, *Edge* tiene $O(C \times M \times N)$, siendo M, N las dimensiones de cada *frame* y C la cantidad de filtros aplicados. De modo que por cada video de longitud L *Edge* tiene una complejidad $O(L \times M \times N)$ (considerando que C es una constante). *Mean Average* tiene una complejidad $O(M \times N)$, siendo M, N las dimensiones de cada *frame* y por tanto para un video de longitud L se tiene $O(L \times M \times N)$. *DCT* tiene orden $O(L \times M^2 \times N^2)$ por cada video, puesto que por cada *frame* calcula tantos coeficientes como dimensiones tenga la imagen, recorriendo cada vez todos los pixeles. *Edge* y *Mean average* tienen complejidades iguales debido a que realizan promedios y cálculos sobre los *frames*, basándose en la lectura de cada uno de los pixeles sólo una vez. *BCS* - sin incluir el cálculo previo del descriptor usado - tiene orden del cálculo del *PCA*, el cual se basa - en OpenCV - en el uso de SVD: *Singular Value Decomposition* y aún sin saber qué método usa OpenCV para resolver esto, no debiera tomar menos de $O(D \times L^2)$ [21]; siendo D la dimensión del descriptor usado y L la cantidad de datos (en este caso la cantidad de *frames*). Así para *frames* de dimensiones $M \times N = 352 \times 288$, un video de longitud $L = 7200$ (5 minutos de duración a una velocidad de 24 fps) y un descriptor como *Edge* de dimensión $D = 9$, se tiene que *DCT* tendría complejidad orden de $6,0 \times 10^{13}$, *Edge* y *Mean average* tendrían complejidad del orden de $7,0 \times 10^9$, mientras que *BCS* sólo un orden $6,0 \times 10^5$.

Cabe notar –como se define en la Sección 3.3–, que BCS utiliza otro descriptor para

generar una descripción del video utilizando dimensión temporal y espacial (mediante análisis de componentes principales). BCS lee de disco los descriptores previamente calculados y luego realiza con ellos *PCA*, sin realizar lecturas de los *frames* del video, lo que acelera mucho el proceso. De este modo, usar BCS sin previamente haber calculado un descriptor es imposible y por tanto uno debiera sumar a las capacidades de procesamiento, uso de CPU y de disco lo que corresponde al cálculo previo del descriptor. Esta suma no debiese cambiar mayormente los valores del descriptor usado, puesto que los valores calculados de BCS son mucho menores; es decir, utilizar BCS junto alguno de los otros descriptores no afecta las capacidad de cálculo de los mismos.

5.4.3. Robustez

Al graficar los datos obtenidos por cada descriptor se obtienen los gráficos de las Figuras 5.8 a 5.12. En ellas se puede apreciar una tendencia del descriptor *Edge* para recuperar los videos relevantes en las primeras 10 posiciones para las transformaciones: *Aspect Ratio, Blur, Captions, Contrast, Encode* y *Subtitles*. Para las transformaciones *Crop, Gamma* y *Mirroring* el promedio de los videos relevantes está entre 20 y 50 y para *Picture in Picture* es arriba de 100. Un promedio para el *ranking* igual 10 es algo considerado robusto puesto que se está trabajando con 12 transformaciones. En líneas generales, *BCS* y *DCT* son competitivos entre sí pero están alejados de *Edge*. *Mean average* es lejos el menos competitivo.

Picture in Picture es una transformación que modifica alrededor del 40 % de cada frame, es por ello que todos estos descriptores globales recuperan en posiciones lejanas los videos relevantes; la transformación resulta ser demasiado "agresiva" y los descriptores pierden su eficacia.

El mejor rendimiento con *Edge*, *DCT* y *Mean average* se tiene para *Encode* y con *BCS* se tiene para *Aspect Ratio*. El peor rendimiento con *Edge, DCT* y *BCS* se tiene para *Picture in Picture*. Sin embargo el peor rendimiento con *Mean average* se tiene para *Mirroring*; esto pues *Mean average* ordena los bloques de cada frame de acuerdo a sus intensidades y al hacer *Mirroring* este orden se revierte y los vectores característicos de ambos se distancian.

Considerando el Cuadro 2.1 se puede inferir que el descriptor *Edge* es indiferente al tipo de transformación aplicada (global/local, espacio de aplicación, reversible/irreversible, aditivas/sustractivas), sin embargo los gráficos sugieren una dependencia a los parámetros

de estas. Esto se aprecia por ejemplo mediante las diferencias encontradas en los resultados para *Gamma* y *Contrast*, en donde ambas transformaciones se aplican en espacios iguales y tienes resultados similares pero con parámetros distintos. Las transformaciones *Pattern*, Texto (*Subtitles*) y *Picture in Picture* son similares de acuerdo al Cuadro 2.1, sin embargo las tres tienen resultados diferentes, lo cual también sugiere una dependencia de los descriptores respecto de los parámetros de "rudeza" en las transformaciones.

El gráfico de la Figura 5.12, muestra un promedio por transformaciones, lo que da una robustez promedio de cada uno de los descriptores. En este caso el orden de robustez es *Edge*, *BCS*, *DCT* y *Mean average*, siendo *Edge* el de mejor rendimiento (promedio para todas las transformaciones alrededor de 30). *BCS* y *DCT* son competitivos entre sí pero no contra *Edge* y por último *Mean average* con un bajísimo rendimiento. Todos los descriptores tienen intervalos de confianza pequeños, es decir poca variabilidad de los resultados para distintos videos, lo que habla de una robustez por cada transformación.

5.4.4. Eficacia

Finalmente el gráfico para las curvas *Precision vs Recall* de la Figura 5.13 muestra -para el caso de *Edge*- una recuperación casi constante y centrada en 0.9 de los videos relevantes, lo que se mantiene para búsquedas con densidades de videos relevantes en el intervalo [0 %, 70 %], para luego bajar rápidamente a una tasa de aproximadamente 5 videos relevantes por cada encontrado. Se puede concluir que *Edge* es -en promedio- el más eficaz a la hora de obtener todos los videos transformados de un video dado. Los descriptores *DCT* y *BCS* tienen una tendencia similar en eficacia (al igual que en robustez), en este caso ambos tienden a disminuir linealmente y la *Precision* para distintos valores de *Recall* es similar en ambos casos. *Mean average* tiene el peor comportamiento y no alcanza a superar una *Precision* del 20 % para todo el rango de *Recall*.

Con los resultados obtenidos en la Figura 5.13 se creó la Tabla 5.7, en donde se ha simulado un escenario posible de búsqueda. En éste se considera una colección de videos C de tamaño c, en la cual para cada video $v \in C$ existen 30 videos transformados. Por cada búsqueda para un video original de consulta (sin transformación), existen 30 videos relevantes. La primera columna del cuadro muestra distintas cantidades de resultados relevantes que se quieren obtener en los resultados, y cada una de las columnas siguientes muestra el tamaño

	Edge	**BCS**	**DCT**	**Mean**
1	2	2	2	9
3	4	4	4	60
6	7	7	8	86
9	10	10	12	129
12	13	14	19	150
15	16	18	26	188
18	19	29	55	200
21	23	44	92	234
24	27	64	104	267
27	39	96	169	386
30	**54**	**250**	**215**	**600**

Cuadro 5.7: Utilizando los resultados de la Figura 5.13, se ha propuesto un escenario en donde el sub-conjunto de videos relevantes tiene un tamaño de $n = 30$ sobre una colección de videos con tamaño $c \gg n$. La primera columna corresponde a la cantidad de resultados relevantes que se quiere obtener en una consulta y las demás columnas muestran cuántos resultados se deberán revisar para obtener dicha cantidad de aciertos.

del conjunto de resultados necesario para tales efectos.

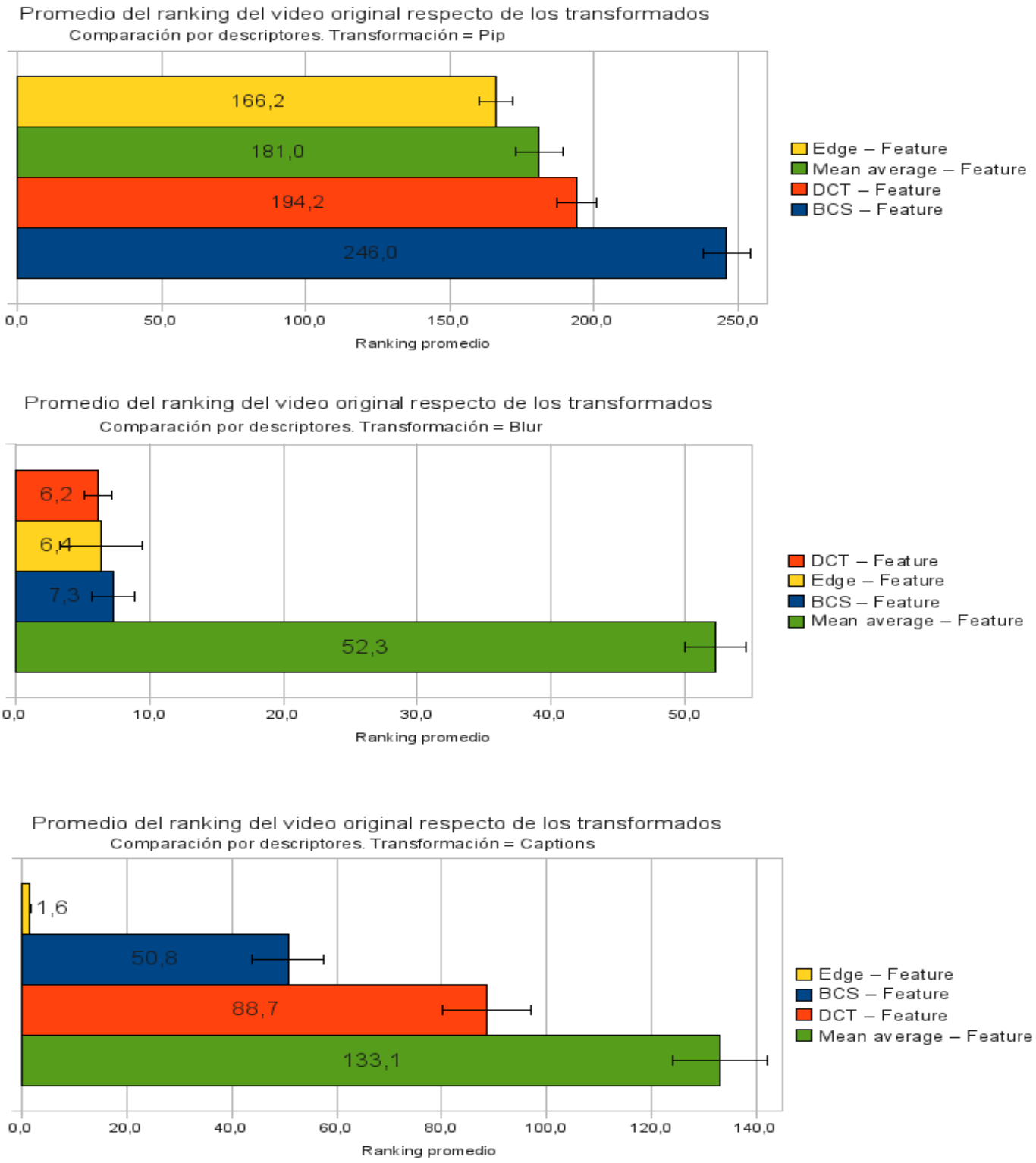

Figura 5.8: Gráfico comparativo de los *ranking* promedio de los videos relevantes para cada uno de los descriptores. Transformaciones:= *Pip*, *Blur* y *Captions*.

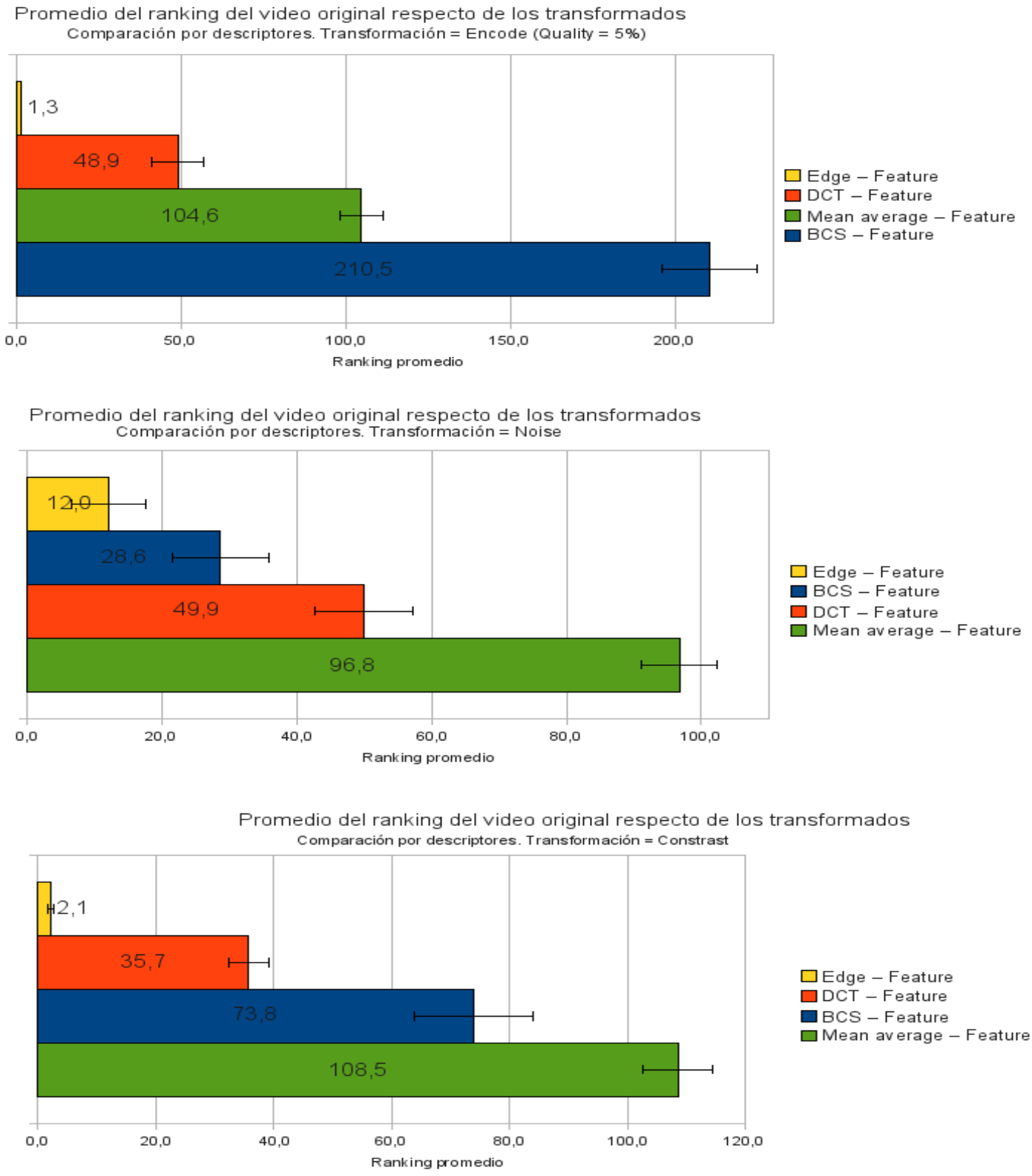

Figura 5.9: Gráfico comparativo de los *ranking* promedio de los videos relevantes para cada uno de los descriptores. Transformación:= *Encode*, *Noise* y *Contrast*.

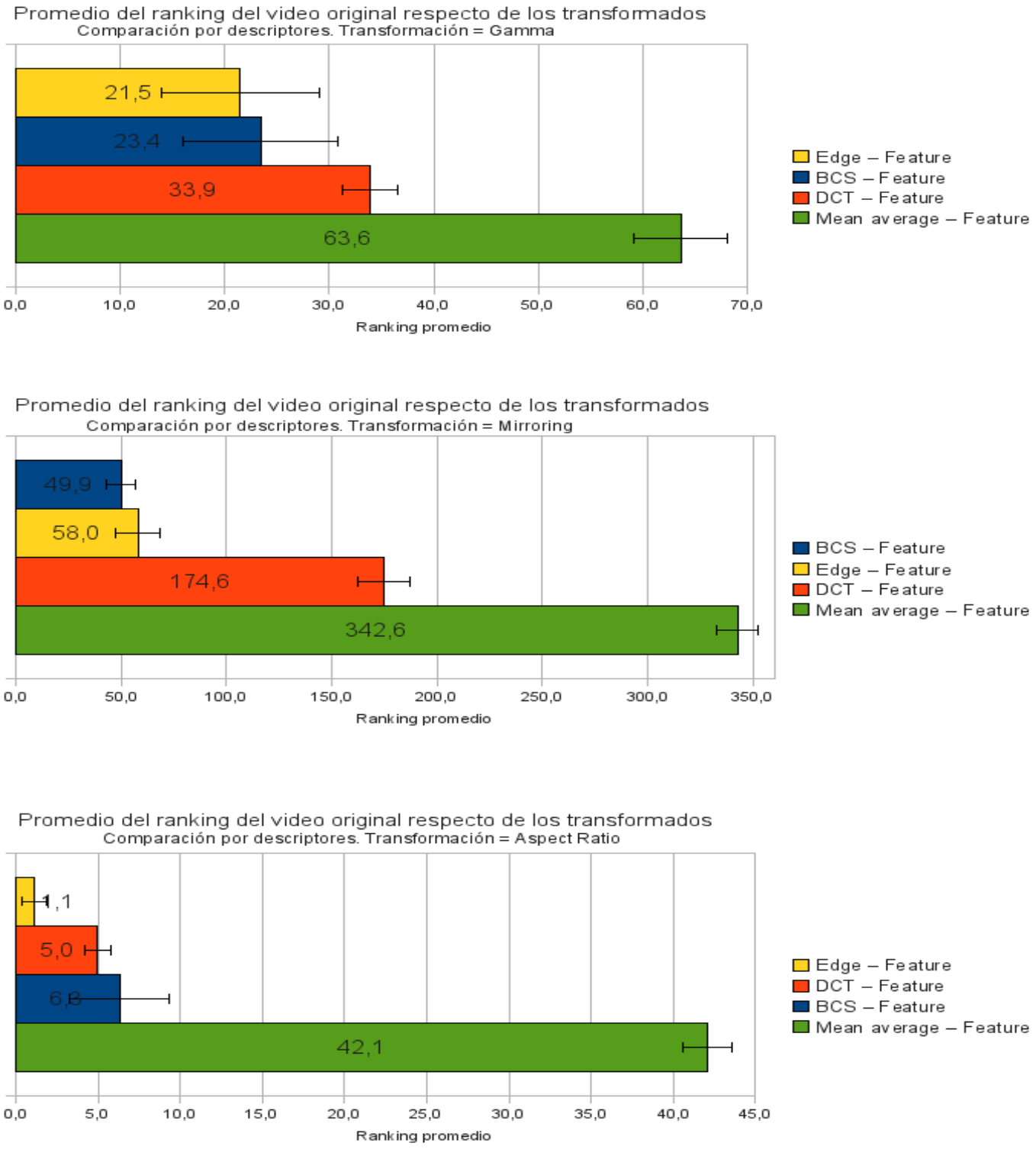

Figura 5.10: Gráfico comparativo de los *ranking* promedio de los videos relevantes para cada uno de los descriptores. Transformación:=*Gamma*, *Mirroring* y *Aspect ratio*.

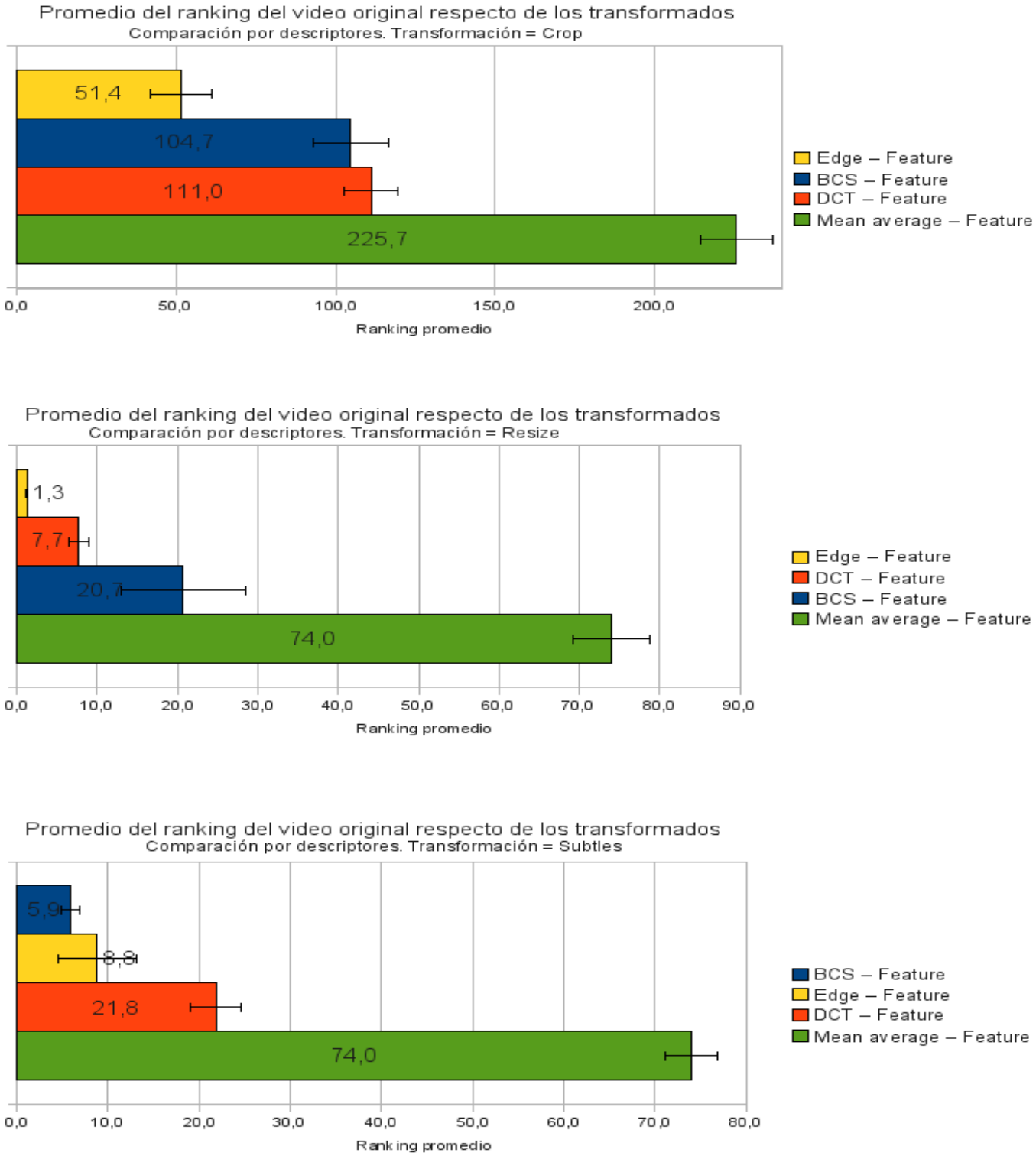

Figura 5.11: Gráfico comparativo de los *ranking* promedio de los videos relevantes para cada uno de los descriptores. Transformación:= *Crop*, *Resize* y *Subtles*.

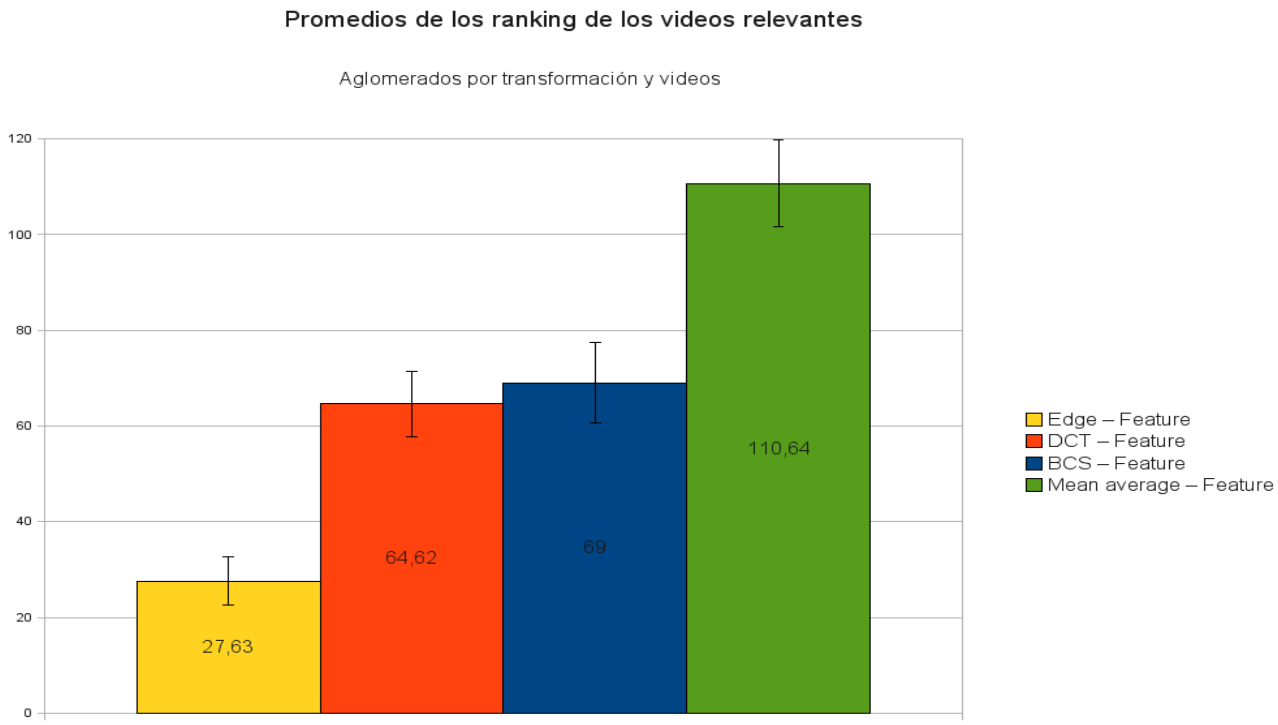

Figura 5.12: Gráfico comparativo de los *ranking* promedio de los videos relevantes para cada uno de los descriptores, aglomerado por transformaciones y videos.

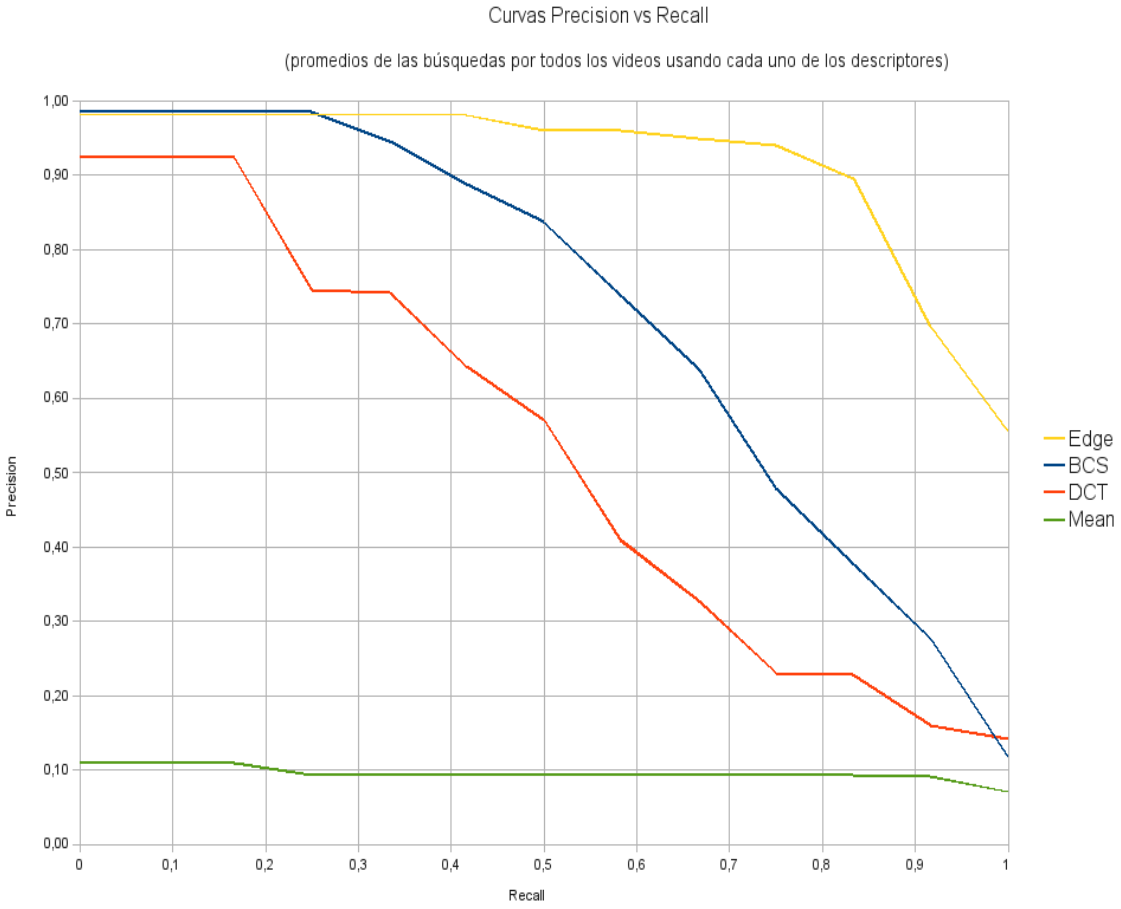

Figura 5.13: Gráfico para curvas Precision-Recall para los cuatro descriptores.

Capítulo 6

Conclusiones

6.1. Conclusiones

A continuación se detallan las conclusiones en cuanto a eficiencia, correctitud, robustez y eficacia. Cada sección contiene un cuadro resumen del rendimiento de cada descriptor, basado en las medidas de estas propiedades en el Capítulo 5.

6.1.1. Eficiencia

BCS, *Edge* y *Mean average* tienen una tasa de cálculo (*fps*) superior a la de reproducción de los videos (24 *fps*). Sin embargo de todos los descriptores, *BCS* es el único independiente del tamaño de los *frames*; en este caso, depende de la dimensión de los descriptores utilizados. La rapidez en el cálculo de *Edge*, *Mean average* y *DCT* dependen tanto de los *frames* de entrada como de los parámetros de ajustes de estos descriptores.

Suponiendo una medida de eficiencia tal que, descriptores con tasas de cálculo $\geq$ 30 fps son considerados eficientes, se construye el Cuadro 6.1 a modo de resumen de eficiencia basado en la Sección 5.4.

	Edge	**DCT-based**	**BCS**	**Mean Average**
Eficiente	✓	×	✓	✓

Cuadro 6.1: Tabla comparativa que muestra cuáles descriptores son eficientes y cuáles no, utilizando como medida comparativa una tasa de cálculo $\geq$ 30 fps.

Edge descriptor

La rapidez de cálculo en *Edge* está relacionada con el tamaño en pixeles de cada *frame*, la cantidad de bloques en los que se divide cada *frame* y la cantidad de filtros aplicados a cada bloque. Mientras mayor sea el tamaño de cada *frame*, más grande serán los bloques (manteniendo la cantidad de ellos fija) y por ende cada sub-bloque tendrá más pixeles que leer de la imagen para calcular su promedio, haciendo que la tasa de cálculo disminuya. Si se mantiene el tamaño de los *frames* constante y se aumenta la cantidad de bloques, cada sub-bloque tendrá menos pixeles que sumar en el promedio, pero habrán más sub-bloques que calcular y por tanto se tendrán que aplicar más filtros, haciendo que la tasa de cálculo disminuya. Si se aumentan los filtros y se mantienen los demás parámetros constantes, habrán más multiplicaciones por cada bloque y aunque la cantidad de éstas sea menor respecto a la cantidad de promedios por cada sub-bloque, bien podría disminuir la tasa de cálculo. El peor caso para la tasa de cálculo entregada en la Tabla 5.2 de este descriptor es el aumento del tamaño de los *frames*, aumento en la cantidad de filtros aplicados y aumento en la cantidad de bloques a dividir cada *frame*.

Una optimización importante de este descriptor se lograría utilizando una unidad de procesamiento gráfico (GPU); estas unidades de procesamiento permiten la paralelización de filtros de *kernel* sobre imágenes, disminuyendo considerablemente el cálculo de los mismos e incluso alcanzando un *speed-up* de 22.5 [4]. De todas formas, una paralelización de cada bloque ya sería una optimización que aumentaría considerablemente la tasa de cálculo.

Mean average descriptor

La tasa de cálculo de *Mean average* depende de la cantidad de bloques en que se divida cada *frame* y del tamaño de cada *frame*. Si existen más bloques - dejando el tamaño de los *frames* constante - se tendrán que calcular más promedios de intensidades, pero la cantidad de pixeles por cada bloque disminuye, por lo tanto la tasa de cálculo debería mantenerse constante o disminuir levemente; esto pues la cantidad de sumandos es la misma, sólo que se han distribuido en distintas sumatorias. Aumentar extremadamente la cantidad de bloques sí afectará el ordenamiento de los mismos y aunque en este caso los *frames* son pequeños, en imágenes muy grandes con cantidad de bloques altas podría afectar la tasa de cálculo. El ordenamiento de los bloques tiene complejidad $O(k \log k)$ –siendo k la cantidad de bloques–

, de modo que a mayor cantidad de bloques, mayor complejidad en el ordenamiento. Sin embargo, el tamaño de cada *frame* sí afecta la tasa de cálculo; al aumentar este tamaño - dejando la cantidad de bloques constante - aumenta la cantidad de pixeles en cada bloque y por tanto el cálculo de cada promedio se hace más lento, lo que tiende a disminuir la tasa de cálculo. El peor caso para la tasa de cálculo entregada como referencia en el Cuadro 5.2, es el aumento en el tamaño de los *frames* y la cantidad de bloques.

Una optimización posible es calcular cada promedio de los bloques de forma paralela y luego ordenar, esto pues cada promedio es independiente de los otros (no así el ordenamiento).

DCT descriptor

Para el descriptor *DCT* la tasa de cálculo depende sólo del tamaño de los *frames*. Al aumentar el tamaño de estos, los sumandos en la Ecuación 3.11 aumentan y por tanto la tasa de cálculo disminuye. La gran diferencia entre este descriptor, *Edge* y *Mean average* está en el cálculo de los cosenos en la Ecuación 3.11 y la cantidad de promedios que realiza por cada *frame*. *Edge* y *Mean average* se parecen mucho entre sí al sólo efectuar un promedio de las intensidades de los *frames* por regiones, sin embargo *DCT* además de efectuar un promedio por regiones, debe calcular por cada sumando dos cosenos y luego ponderarlos por la intensidad. Además *DCT* por cada coeficiente que calcula, efectúa un promedio sobre todos los pixeles del *frame*, lo que induce el orden cuadrático en las dimensiones de los *frames*. El peor caso para la tasa de cálculo entregada como referencia en el Cuadro 5.2 es el aumento del tamaño de los *frames*.

Una forma de optimizar este cálculo, es implementando una clase para el cálculo de la DCT (Ecuación 3.11) utilizando programación dinámica, para calcular los cosenos en el primer *frame* y luego guardarlos y utilizarlos sucesivamente en los siguientes (dado que éstos son constantes para cada coeficiente y cada *frame*). Además esta clase debería considerar el cálculo de $n < N \times M$ coeficientes (siendo M, N las dimensiones de los *frames*) y no todos los coeficientes, como lo hace actualmente la librería de OpenCV. También se pueden paralelizar cálculos para este descriptor, especialmente para los coeficientes de cada *DCT*; puesto que éstos son independientes de los anteriores y de los sucesivos.

BCS descriptor

No depende de los *frames* de entrada sino sólo de la dimensión del descriptor usado como entrada para este algoritmo. Las imágenes en *OpenCV* se cargan en memoria para luego ser accesadas mediante una clase del tipo interfaz, de modo que la lectura a disco se ejecuta sólo una vez. Sin embargo, las imágenes son matrices de tamaño $3 \times M \times N$ (siendo M, N las dimensiones de la imagen y 3 la dimensión del espacio de colores), de modo que cada lectura de un frame por cada uno de los descriptores *Edge*, *Mean average* y *DCT* involucra al menos $M \times N$ accesos a memoria (considerando que sólo utilizan una dimensión del espacio de colores; a saber la de intensidad). Por su parte *BCS* tiene que efectuar sólo d lecturas a disco por cada *frame*, y en este caso $d \ll M \times N$. Para un video de largo S, *Edge* y *Mean average* efectúan al menos $S \times M \times N$ lecturas y operaciones; *BCS* efectúa $S \times d^2$ operaciones. De este modo aún cuando *BCS* sea un método basado en *PCA* (el cual incluye cálculo de los vectores y valores propios de una matriz), sigue siendo más eficiente que *Edge* y *Mean average* mientras la dimensión de los vectores sea menor comparada con las dimensiones de los *frames*. De este modo el peor caso para la tasa de cálculo entregada en el Cuadro 5.2 es el aumento de la dimensión de los descriptores sobre los que opera *BCS* a niveles cercanos a los de las dimensiones de los $frames$.

No hay grandes optimizaciones que se puedan hacer a la implementación de este descriptor, puesto que para realizar PCA, existen algoritmos altamente estudiados.

Cabe destacar que para cuantificar los cambios en las tasas de cálculo respecto a los parámetros de cada descriptor y los tamaños de las imágenes, se necesita efectuar un análisis de sensibilidad de dichos parámetros, en donde se registre el tiempo tomado por cada descriptor en realizar los cálculos utilizando distintos escenarios.

6.1.2. Correctitud

Durante el desarrollo de los experimentos de correctitud, se detectó que sólo *DCT* estaba implementado correctamente; los demás descriptores debieron pasar por un etapa de detección de *bugs* y volver a correr los experimentos de correctitud. Considerando que los experimentos siguientes tomaron un buen tiempo, determinar la correctitud de los descriptores fue fundamental para el cumplimiento de los plazos planteados; de no haber dedicado un tiempo al diseño y ejecución para comprobar la correctitud, el tiempo final de los experi-

	Edge	DCT-based	BCS	Mean Average
Correcto	✓	✓	✓	✓

Cuadro 6.2: Tabla comparativa que muestra cuáles descriptores son correctos y cuáles no, utilizando como medida comparativa el hecho de aprobar los experimentos de la Sección 5.2.

mentos siguientes se podría haber - en el mejor caso - duplicado e incluso los errores habrían pasado desapercibidos, terminando en resultados erróneos para robustez y eficacia.

Suponiendo que un descriptor es correcto si y sólo si pasa los experimentos aplicados en la Sección 5.2, se presenta el Cuadro 6.2.

6.1.3. Robustez

Sea:

- Un video de consulta V_q.
- Una colección de videos C de tamaño $N \times T$, siendo $N = 30$ un sub-conjunto de C con los videos originales -no transformados- y $T = 12$ un conjunto de transformaciones sobre los videos.

Al utilizar búsqueda exhaustiva para encontrar todos los videos transformados de V_q en C utilizando *Edge*, se obtiene un promedio de las posiciones en el *ranking* de 20. Los descriptores *BCS*, *DCT* obtienen los mismos videos en un *ranking* promedio de 60 . *Mean average* los obtiene en promedio en los primeros 110 resultados.

A continuación se concluirá respecto de los resultados obtenidos por cada uno de los descriptores.

Edge descriptor

Se puede apreciar cómo éste descriptor es insensible a cambios globales y locales en las intensidades sobre las áreas de los *frames* que evalúa y se basa en un nivel de umbral para catalogar los distintos bloques en alguno de los 11 gradientes de intensidades. De este modo cambios en la entrada como *Blur*, *Contrast* y *Encode* son fáciles de detectar mediante este descriptor y sus resultados en búsquedas se ubican en los primeros 10 resultados; *Blur* y *Encode* se basan en promedios de intensidades para modificar la información de entrada y

Contrast sólo resalta las diferencias de intensidades, haciendo más fácil aún encontrar el video en los primeros lugares (notar en la Figura 5.9 cómo *Contrast* es uno de los que tiene mejores resultados). *Noise* es otra transformación similar a las anteriores, en este caso se modifican algunos pixeles de los *frames*, pero como se agregan de forma regular sobre éstos, no influye mayormente en el promedio de intensidades. *Gamma* tiene resultados dentro de los 20 primeros, haciendo de las transformaciones que manejan intensidades la de peor robustez, debido a que ésta no conserva los gradientes de intensidades y tiende a aplanar la imagen en cuanto a intensidades (esta transformación se basa en exponenciar cada intensidad de la imagen, tal como se explicó en la Sección 2.2.2).

Por otro lado, al revisar la Ecuación 3.2, se entiende porqué este descriptor es robusto frente a transformaciones como *Captions* y *Subtles*. Estas transformaciones modifican sólo una parte de los *frames* (una región rectangular acotada) y por ende modificarán la clasificación de unos pocos bloques, pero los demás serán clasificados igual que el original y por tanto la diferencia -entre video original y transformado- será de por ejemplo $\frac{p}{K^2}$ bloques por cada *frame*, lo que sin duda no es mucho mientras $p \ll K^2$ (en este caso $p \sim 1$).

Edge es robusto a transformaciones de tamaño como *Resize* y *Aspect ratio*, puesto que éstas no alteran la información de las intensidades promedios de cada cuadrante y los pixeles que caen dentro de un promedio en un bloque determinado seguirán siendo los mismos al expandir el tamaño de la imagen. El parámetro T (umbral) para este descriptor es importante al permitir que algunos bloques se consideren como no clasificados ($x_i = 11$) y no sean considerados en la Ecuación 3.2 (en donde se impuso que $x_i = 11$ o $y_i = 11$ implica $E(x_i, y_i) = 0$), permitiendo que no se comparen bloques de los cuales no se puede asegurar su clasificación.

Transformaciones como *Crop*, *Mirroring* y *Pip* son más difíciles de detectar por este descriptor. En el caso de *Crop* sucede que al quitar información de los *frames*, el promedio de intensidades se ve fuertemente afectado no sólo por la falta de información si no también por la división de los cuadrantes (los cuadrantes se han desplazado dentro de la imagen, haciendo que los sumandos de intensidades cambien). Para el caso de *Mirroring* sucede algo similar, puesto que los sumandos de cada bloque han cambiado completamente y la comparación entre videos respeta la localidad de los bloques (el bloque del video q ubicado en $(0, 0)$ se compara ahora con el mismo bloque del video transformado, pero éste bloque corresponde al $(M, 0)$ en el video original).

El caso de *Pip* es un poco distinto y es que esta transformación encuadra el video original sobre un video cualquiera; de este modo el video original pasará a ser del tamaño de un cuadrante (o un par quizás) y por ende no coincidirán en nada al comparar con el original.

De esta forma *Edge* es robusto a cambios locales en el espacio de intensidades, cambios locales y globales en el espacio de colores, cambios de tamaño y transformaciones aditivas.

DCT descriptor

Depende de las intensidades de los pixeles en los *frames* de forma ponderada (I en la Ecuación 3.11) y del tamaño de los mismos (cosenos y denominador $\sqrt{2MN}$ en la Ecuación 3.11). A pesar de que DCT también efectúa un promedio ponderado, ésta lo hace sobre toda la imagen y podría pensarse que es más robusto a pequeños cambios en las intensidades, sin embargo, cada uno de los coeficientes de este descriptor promedia las intensidades de todos los pixeles de los *frames* lo que produce un comportamiento menos robusto que *Edge*. Esto se debe a que *Edge* opera ponderando por bloques en la imagen y luego compara dichos bloques, por tanto todas las transformaciones que modifiquen sólo una sección de la imagen, no tendrán grandes consecuencias en el descriptor en general (sino sólo en un par de bloques del mismo). Al aplicar promedios ponderados sobre toda la imagen, DCT es menos robusto que *Edge* en transformaciones como *Captions* y *Subtles*, pues en donde *Edge* se ve afectado en algunos bloques, DCT se ve afectado en todos sus coeficientes.

En transformaciones como *Blur*, *Resize* y *Aspect ratio*, DCT es competitivo frente a *Edge*. *Blur* promedia intensidades en regiones acotadas en cada uno de los pixeles y por ende no tiene gran impacto en el promedio ponderado general del *frame* que hace DCT. Por otro lado los cambios en el tamaño de la imagen que hace *Resize* tienen influencia sólo en las operaciones de coseno de la Ecuación 3.11, pero basta con calcular dichos términos variando M y N, para notar que la diferencia es del orden de 10^{-4} (por cada término coseno).

Para *Encode* y *Noise*, este descriptor es menos robusto puesto que guarda directamente un valor basado en intensidad (aún cuando esté cuantizado) y *Edge* por su parte, compara clasificaciones basadas en un umbral, permitiendo mayores diferencias en las intensidades. El caso de *Mirroring* (donde el descriptor se desempeña mal en cuanto a robustez), se debe a que si bien esta transformación no modifica las intensidades, sí modifica los ponderadores de las mismas (cosenos en Ecuación 3.11), de modo que los coeficientes se ven altamente

afectados.

Para la transformación *Pip*, *DCT* tiene un comportamiento similar al de los demás descriptores y su bajo rendimiento se debe a que la imagen que se intenta encontrar es una parte pequeña dentro de la imagen que se está evaluando, de modo que los coeficientes encontrados no son representativos del video original. Por su parte *Crop* modifica no sólo los cosenos (al cambiar las intensidades para cada una de las posiciones originales) sino que también la intensidad de los pixeles y de allí su mal rendimiento (el promedio del *ranking* de los videos buscados casi dobla al de *Edge*). En cuanto a *Contrast*, al ponderar las intensidades y llevarlas a coeficientes, tiene peores resultados que *Edge*, puesto que *Edge* se basa en las intensidades para clasificar los bloques de acuerdo a un gradiente de intensidades (el gradiente en *Contrast* se mantiene pero en *DCT* los coeficientes cambian al cambiar las intensidades). En el caso de *Gamma* el descriptor *DCT* cambia los valores de Intensidad de los pixeles, modificando los coeficientes de todo el descriptor.

Este descriptor en general es más robusto a cambios en intensidades locales sobre espacios de colores y que no afecten el tamaño de los *frames*.

Mean average descriptor

Depende de la intensidad promedio de los pixeles del *frame* divididos en bloques, sin embargo este descriptor no guarda información respecto de las intensidades y la posición que utilizan en los *frames* (como sí lo hace *Edge*). *Mean Average* guarda el orden relativo (por intensidades) de los bloques y finalmente lo que compara en la Ecuación 3.3 es diferencias en el ordenamiento de los bloques. De este modo es muy fácil que un video obtenga el mismo ordenamiento de los bloques pero con intensidades totalmente distintas. Si un bloque cambia radicalmente su intensidad entonces se tendrán dos bloques con ordenamientos distintos, si cambian dos bloques se tendrán cuatro bloques con órdenes distintos y en general si se cambian n bloques de intensidad se tendrán $2 \times n$ cambios en los componentes del descriptor.

De este modo transformaciones como *Subtles* y *Captions* tienen comportamientos similares debido a que modifican partes importantes de los *frames* y si bien no abarcan varios bloques, el factor duplicador en el orden introduce cambios sustanciales. *Subtles* modifica levemente las intensidades en la medida que los títulos estén acotados a la menor cantidad de bloques; mientras más grande sean los bloques, menos afectarán las intensidades de los

subtítulos al promedio de los mismos. En el caso de *Captions* el área modificada de la imagen abarca uno o dos bloques, que se traduce en al menos 4 dimensiones distintas entre el descriptor original y el transformado; estos bloques se propagan por todo el video, modificando además la dimensión temporal de este descriptor.

Los mejores *set* de resultados se obtienen con las transformaciones: *Gamma*, *Noise* y *Blur* dado que introducen modificaciones pequeñas que no afectan el promedio de cada bloque y permiten mantener el orden de los mismos. Con similar resultado están *Aspect ratio* y *Resize* que modifican levemente las intensidades.

Mirroring es la transformación en la que se obtienen los peores resultados incluso comparado con los demás descriptores. Esto se debe a que en este caso se invierte el orden de los bloques (tal como se invierte la imagen). En el caso de *Crop* sucede que el corte de información afecta a muchos bloques (puesto que crop se aplica sobre toda una dimensión, ya sea de alto o de ancho).

Este descriptor es altamente sensible a cambios locales y globales en el espacio de intensidades, tamaño de la imagen y en general cualquier transformación sobre un bloque tiene un efecto duplicador.

BCS

Tiene una robustez promedio similar a la de *DCT*. Al basarse en un descriptor robusto como *Edge*, tiene altas expectativas para ser robusto, sin embargo ocurre que *BCS* se basa en reducción de las dimensiones de la matriz que describe el video. Esta reducción de información es agresiva, puesto que se pasa de una matriz de $L \times d$ - con L el largo del video y d la dimensión del descriptor utilizado con *BCS*- a una de $(d+1) \times d$ (d vectores característicos de dimensión d más un vector que contiene el promedio de las dimensiones). Así al utilizar el descritor *BCS* sobre un descriptor *Edge* de dimensión $d = 10$, se tienen dimensiones de orden $L \times 10$ para *Edge* y $11 \times 10 = 110$ para *BCS*, con $L \gg 10$ (en general $L \sim 1,0 \times 10^3$). Si bien esta reducción de información permite mejores tiempos en la búsqueda exhaustiva, también disminuye la capacidad inherente de *Edge* para encontrar los videos transformados en los primeros lugares de los resultados.

Cabe recordar que *Edge*, para comparar dos videos, utiliza una función de similitud (que no es distancia), en la cual sólo se comparan aquellas dimensiones x_i en donde $x_i \neq 11$, mien-

	Edge	**DCT-based**	**BCS**	**Mean Average**
Pip	×	×	×	×
Blur	✓	✓	✓	×
Captions	✓	×	×	×
Encode	✓	×	×	×
Noise	✓	×	×	×
Contrast	✓	×	×	×
Gamma	×	×	×	×
Mirroring	×	×	×	×
Aspect ratio	✓	✓	✓	✓
Crop	×	×	×	×
Rezise	✓	✓	✓	×
Subtles	✓	✓	×	×

Cuadro 6.3: Tabla comparativa que muestra cuáles descriptores son robustos y frente a qué transformaciones.

	Edge	**DCT-based**	**BCS**	**Mean Average**
Nivel de robustez	66 %	33 %	25 %	8 %
Robusto	✓	×	×	×

Cuadro 6.4: Tabla comparativa que muestra el nivel de robustez de los descriptores, utilizando como medida la cantidad de transformaciones en las que el descriptor es robusto del total de transformaciones.

tras que en *BCS* se utiliza la función de distancia euclidiana en donde todas las dimensiones son comparadas mediante diferencias. En *Edge* cada dimensión representa una clasificación y no una posición, de modo que comparar las dimensiones mediante distancias -como la euclidiana- pierde sentido. *Edge* perfectamente podría clasificar cada bloque con una letra o un símbolo, en cuyo caso la distancia Euclidiana pierde todo sentido.

Observando los gráficos de las Figuras 5.8 a 5.11 y considerando una medida cuantitativa de robustez tal que, un descriptor se dice robusto a una transformación si y sólo el video transformado buscado se encuentra dentro de los primeros 20 resultados; se presenta entonces el Cuadro 6.3. Así mismo, se genera una medida relativa de robustez definida como el cuociente entre número de transformaciones en las que el descriptor es robusto sobre el total de trasnformaciones, presentadas en forma de porcentajes en el Cuadro 6.4. En este cuadro, un descriptor se dice robusto si su nivel de robustez es $\geq 50\,\%$.

	Edge	**DCT-based**	**BCS**	**Mean Average**
Eficaz	✓	✓	✓	×

Cuadro 6.5: Tabla comparativa que muestra qué descriptores son eficaces, utilizando como medida una $Precision \geq 50\,\%$ para un $Recall \geq 50\,\%$.

6.1.4. Eficacia

La Figura 5.13 muestra cómo mediante *Edge* se pueden obtener dentro de todo el conjunto de resultados de una búsqueda, un 90 % de videos transformados relevantes, si y sólo si la relación entre videos relevantes encontrados del universo de videos relevantes se encuentra en el rango $[0\,\%, 80\,\%]$. Esto conduce a los resultados del Cuadro 5.7, en donde como resultado general se puede decir que para encontrar todos los videos transformados relevantes (que suman un total de 30), se necesitará un conjunto de resultados de tamaño 54 para *Edge*, 215 para *DCT*, 250 para *BCS* y 600 para *Mean average*. Si queremos sólo la mitad de los videos transformados relevantes, necesitaremos un conjunto de resultados de tamaño 16 para *Edge*, 18 para *BCS*, 26 para *DCT* y 188 para *Mean average*. De modo que *Edge* es al menos dos veces más eficaz que *DCT* y *BCS*. Respecto a *Mean average*, *Edge* es órdenes de magnitud más eficaz. Esto tiene sentido si consideramos las conclusiones de robustez propuestas en el apartado anterior.

Considerando una medida de eficacia tal que: un descriptor es eficaz si logra recuperar al menos el 50 % del total de objetos relevantes de una colección ($Recall \geq 50\,\%$), en donde al menos el 50 % del conjunto de resultados es parte de los objetos relevantes ($Precision \geq 50\,\%$), se genera el Cuadro 6.5.

6.1.5. Consideraciones finales

Como conclusión final se presenta el Cuadro 6.6 en donde se han resumido todas las medidas de las secciones anteriores. El descriptor *Edge* cumple con todas las propiedades deseables de un descriptor: correcto, eficiente, eficaz y robusto. *BCS* no es robusto pero sí correcto, eficiente y eficaz. *DCT* es sólo correcto y eficaz. *Mean average* es sólo correcto y eficiente.

Considerando los resultados obtenidos, junto con el respectivo profundo estudio de cada uno de los descriptores, se sugieren las siguientes formas de operar de un descriptor:

	Edge	**DCT-based**	**BCS**	**Mean Average**
Correcto	✓	✓	✓	✓
Eficiente	✓	×	✓	✓
Eficaz	✓	✓	✓	×
Robusto	✓	×	×	×

Cuadro 6.6: Tabla comparativa con un resumen final de las propiedades que cumplen cada uno de los descriptores.

1. División en bloques de las imágenes, permitiendo robustez y eficacia frente a transformaciones locales y manteniedo información espacial de los *frames* mediante comparación bloque por bloque en las funciones de distancia o similitud.

2. El uso de funciones matemáticas y parámetros estadísticos que resuman la información en el espacio de intensidades, en especial se recomienda el uso de momentos centrados (promedios, varianzas, curtósis, etc.). Esto permite robustez y eficacia frente a cambios globales en el espacio de intensidades.

3. La información de los vectores debe estar basada en propiedades de los bloques y no en relaciones entre ellos, cada vector debiera acarrear en alguna dimensión, alguna información basada en las intensidades de los mismos. Esto permite eficacia al comparar la verdadera información que compone la imagen.

4. Descriptores con parámetros de ajustes, permitiendo sensibilizar los resultados.

Finalmente y sólo como conclusión al margen de esta memoria, cabe destacar a la búsqueda exhaustiva como un proceso lento y que consume muchos recursos. En este trabajo no se ha considerado la medición de tiempos y recursos para la etapa de búsqueda, sin embargo una de las experiencias que este proceso dejó, es que al menos la búsqueda exhaustiva es quizá dos o incluso tres veces más lenta que la obtención de los descriptores. Por otro lado, la búsqueda exhaustiva tiene la limitación de no permitir buscar segmentos de video dentro de videos. Sea el caso en que se tiene un video de larga duración (más de 24 horas), para buscar si existe algún segmento de alguno de los videos de una colección, búsqueda exhaustiva tomaría un tiempo extremadamente largo. De hecho, sólo el número de segmentos de consulta es $O(n^2)$, con n el número de *frames* del video de consulta.

6.2. Trabajo futuro

A continuación se listan algunas proposiciones para un trabajo futuro:

- Análisis de sensibilidad de la eficacia y robustez de los descriptores en relación a los parámetros de cada uno los descriptores.
- Análisis de sensibilidad de la eficacia y robustez de los descriptores en relación a las funciones de distancia (o similitud en el caso de *Edge*).
- Análisis de sensibilidad de la eficacia y robustez de los descriptores en relación a la colección de videos utilizados en las búsquedas.
- Análisis de sensibilidad de la eficacia y robustez de los descriptores en relación al tamaño de la colección de videos.
- Optimización de los descriptores para disminuir el tiempo de cálculo (implementar paralelismo y uso de GPU).
- Estudio de alternativas a búsqueda exhaustiva utilizando índices en cada uno de los descriptores.

Apéndices

A . Figuras

B . Códigos

B .1. Consultas Sql

Ranking **de distancias**

```
 SELECT @ranking:=@ranking+1 AS ranking, object, objectTransformation, query, "+
"queryTransformation, descriptor, distance from distancesString "+
"WHERE object = '"+thisMovie+
"' AND objectTransformation='"+thisTransformation+
"' AND descriptor = '"+thisDescriptor+"' "+
"ORDER BY distance";



SET @ranking=0



"INSERT INTO rankingDistances(object, objectTransformation, query, "+
"queryTransformation, descriptor, distance, ranking) VALUES('"+
object+"','"+ objectTransformation +"','"+ query +"','"+
queryTransformation +"','"+ descriptor+"',"+ distance+","+ rank+")";
```

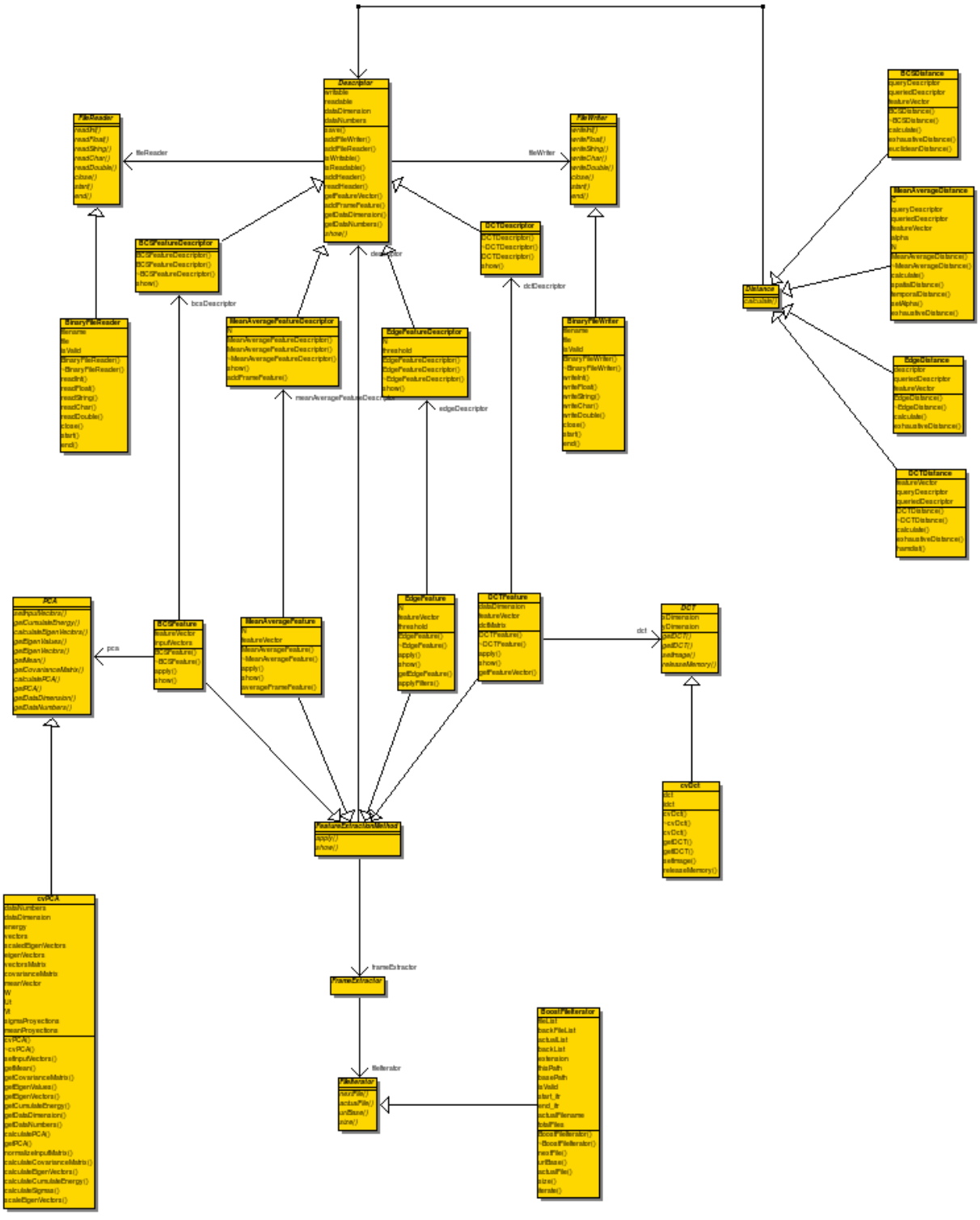

Figura 6.1: Diagrama de clases de la implementación. En este caso se utilizó un esquema base que fue modificado mientras se implementaba. Finalmente se creó el esquema de forma reversa, es decir a partir del análisis de la implementación en C++ mediante el programa para diagramar BoUml.

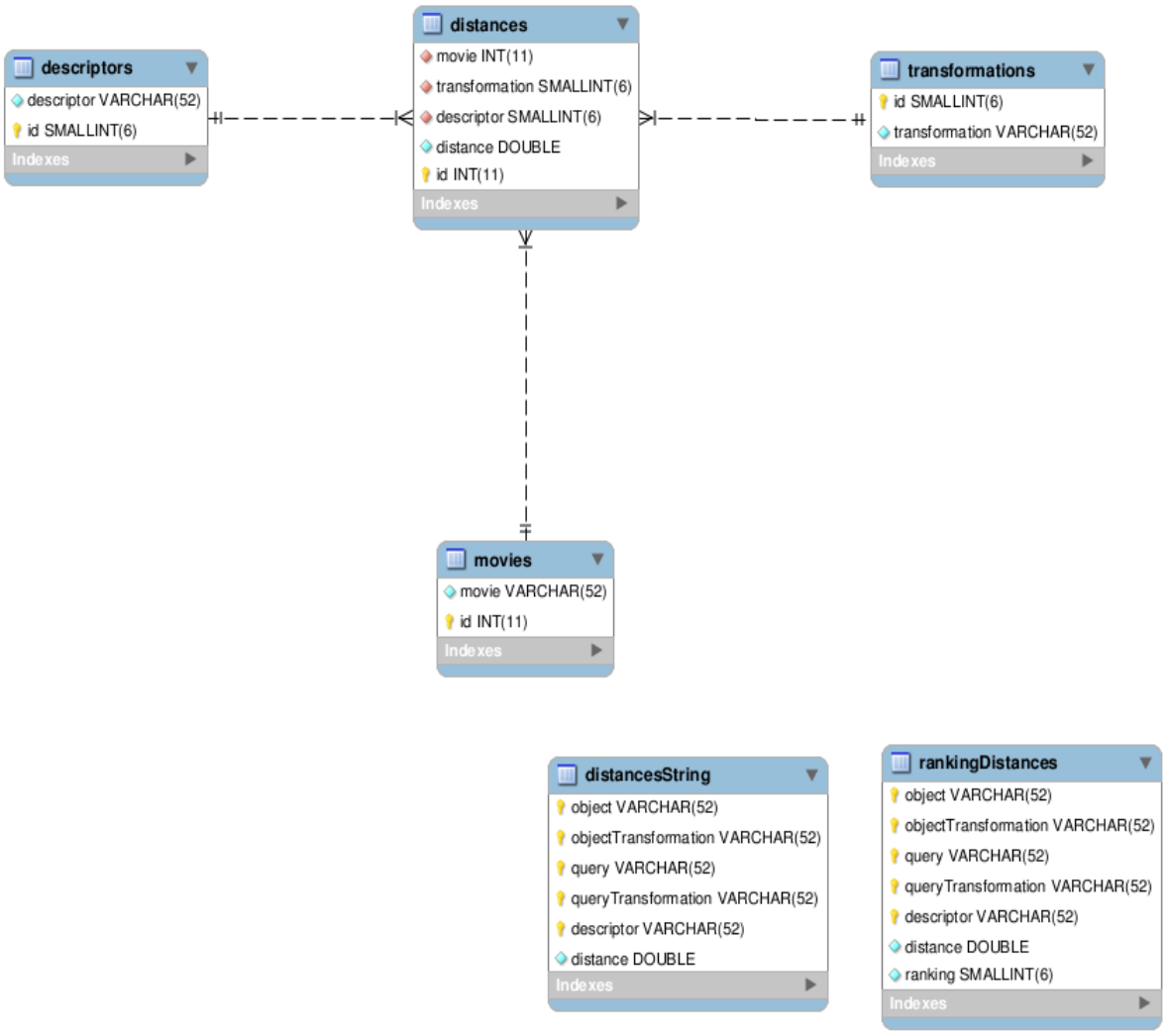

Figura 6.2: Esquema de la base de datos relacional para resumir los resultados obtenidos para las distancias entre todos los videos. No se utilizaron índices ni llaves foráneas, debido a que las agregarlos complejizaba las consultas y no aportaba mayor eficiencia considerando que la cantidad de consultas sobre la base de datos era mínima (alrededor de 10).

Consultas para obtención de resultados agregados

```
PROMEDIO RP
============

SELECT avg(RP) FROM (SELECT object, count(object) as RP FROM
(SELECT * FROM rankingDistances
 WHERE objectTransformation="NONE"
 AND ranking <= 3 AND object=query AND descriptor like "DCT%"
ORDER BY object, ranking)
as A group BY object ORDER BY object) as B;



PROMEDIOS RANKING VIDEO TRANSFORMADO VERSUS VIDEO ORIGINAL
============================================================

SELECT object, queryTransformation, avg(ranking) FROM
(SELECT object, queryTransformation, ranking FROM rankingDistances
 WHERE objectTransformation="NONE"
 AND object=query AND queryTransformation!="NONE"
 AND descriptor like Edge%
         ORDER BY queryTransformation, ranking) as A
group BY queryTransformation, object ORDER BY object;



PROMEDIO Y DESVIACION ESTANDAR DE LA POSICION EN EL
RANKING DE LOS VIDEOS TRANSFORMADOS DE UN VIDEO
TRANSFORMADO PARTICULAR
=================================================

SELECT object, objectTransformation, AVG(ranking) AS average,
```

```
STDDEV(ranking) AS stddev
INTO OUTFILE 'promediosRanking.csv'
FIELDS TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM rankingDistances
WHERE object=query
AND objectTransformation!=queryTransformation
AND descriptor="EdgeFeatureDescriptor"
GROUP BY object, objectTransformation
ORDER BY average, stddev;
```

ENVIAR EL RESULTSET A UN ARCHIVO CON FORMATO

==========

```
INTO OUTFILE '<nombre_archivo>.csv' FIELDS
TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

TODOS LOS VIDEOS TRANSFORMADOS QUE FUERON RANKEADOS
MENOR QUE 10 RESPECTO AL MISMO VIDEO CON DISTINTA TRANSFORMACION

==========

```
SELECT object, objectTransformation, query, queryTransformation, ranking
FROM rankingDistances
WHERE object=query
```

```
AND objectTransformation!=queryTransformation
AND descriptor="EdgeFeatureDescriptor"
AND ranking<=1
GROUP BY object, objectTransformation, query, queryTransformation
ORDER BY object
```

```
CUENTA CUÁNTOS VIDEOS FUERON RANKEADOS MENOR QUE 10
RESPECTO AL MISMO VIDEO CON DISTINTA TRASNFORMACION
==================================================
```

```
SELECT object, objectTransformation, count(object) FROM
(SELECT object, objectTransformation, query, queryTransformation, ranking
 FROM rankingDistances
 WHERE object=query
 AND objectTransformation!=queryTransformation
         AND descriptor="EdgeFeatureDescriptor"
         AND ranking<=1
 GROUP BY object, objectTransformation, query, queryTransformation
         ORDER BY object) as A
GROUP BY object, objectTransformation;
```

```
RAKING DEL MISMO VIDEO
======================
```

```
SELECT object, objectTransformation, ranking FROM rankingDistances
WHERE object=query
AND objectTransformation=queryTransformation
```

```
AND descriptor="EdgeFeatureDescriptor"
GROUP BY object, objectTransformation, query, queryTransformation
ORDER BY object;
```

```
PROMEDIO Y DESVIACION ESTANDAR DEL RANKING DEL MISMO VIDEO
==========================================================
```

```
SELECT AVG(ranking), STDDEV(ranking) FROM
(SELECT object, objectTransformation, ranking FROM rankingDistances
 WHERE object=query
         AND objectTransformation=queryTransformation
         AND descriptor="EdgeFeatureDescriptor"
         GROUP BY object, objectTransformation, query, queryTransformation
         ORDER BY object) as A;
```

```
PROMEDIO Y DESVIACION ESTANDAR DE LA CANTIDAD DE VIDEOS QUE FUERON
RANKEADOS EN LOS PRIMEROS 10 LUGARES RESPECTO DEL MISMO VIDEO PERO
CON DISTINTA TRANSFORMACION
==================================================================
```

```
SELECT AVG(counter), STDDEV(counter) FROM
(SELECT object, objectTransformation, count(object) AS counter FROM
(SELECT object, objectTransformation, query, queryTransformation, ranking
 FROM rankingDistances
 WHERE object=query
                 AND objectTransformation!=queryTransformation
                 AND descriptor="EdgeFeatureDescriptor"
```

```
        AND ranking<=10
        GROUP BY object, objectTransformation, query, queryTransformation
        ORDER BY object) as A
GROUP BY object, objectTransformation) AS B;
```

Referencias

[1] Y. Aslandogan and C. Yu. Techniques and systems for image and video retrieval. *Knowledge and Data Engineering, IEEE Transactions on*, 11(1):56–63, Jan/Feb 1999.

[2] R. Baeza-Yates and B. Ribeiro-Neto. Modern information retrieval, 1999.

[3] D. G. R. Bradski and A. Kaehler. *Learning opencv, 1st edition*. O'Reilly Media, Inc., 2008.

[4] D. Castaño-Díez, D. Moser, A. Schoenegger, S. Pruggnaller, and A. S. Frangakis. Performance evaluation of image processing algorithms on the gpu. *Journal of Structural Biology*, 164(1):153 – 160, 2008.

[5] A. M. Ferman, A. M. Tekalp, and R. Mehrotra. Robust color histogram descriptors for video segment retrieval and identification. *Image Processing, IEEE Transactions*, 11(5):497– 508, 2002.

[6] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the qbic system. *Computer*, 28(9):23–32, Sep 1995.

[7] E. Gamma, J. Vlissides, R. Johnson, and R. Helm. *Design Patterns CD: Elements of Reusable Object-Oriented Software, (CD-ROM)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[8] J. Huang, Z. Liu, Y. Wang, Y. Chen, and E. Wong. Integration of multimodal features for video scene classification based on hmm. *Multimedia Signal Processing, 1999 IEEE 3rd Workshop on*, pages 53–58, 1999.

[9] R. W. G. HUNT. *Measuring Colour*. Halsted Press, New York, 1989.

[10] K. Iwamoto, E. Kasutani, and A. Yamada. Image signature robust to caption superimposition for video sequence identification. *ICIP '06: International Conference on Image Processing*, pages 3185–3188, 2006.

[11] C. Kim and B. Vasudev. Spatiotemporal sequence matching for efficient video copy detection. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(1):127–132, Jan. 2005.

[12] J. Law-To, O. Buisson, V. Gouet-Brunet, and N. Boujemaa. Robust voting algorithm based on labels of behavior for video copy detection. *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 835–844, 2006.

[13] A. Michelson. *Studies in Optics*. U. of Chicago Press, 1927.

[14] X. Naturel and P. Gros. A fast shot matching strategy for detecting duplicate sequences in a television stream. *CVDB '05: Proceedings of the 2nd international workshop on Computer vision meets databases*, pages 21–27, 2005.

[15] M. Nixon and A. S. Aguado. *Feature Extraction & Image Processing, Second Edition*. Academic Press, 2 edition, January 2008.

[16] C. Poynton. *Digital Video and HDTV Algorithms and Interfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[17] H. T. Shen, X. Zhou, Z. Huang, J. Shao, and X. Zhou. UQLIPS: a real-time near-duplicate video clip detection system. *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1374–1377, 2007.

[18] A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(12):1349–1380, Dec 2000.

[19] B. Stroustrup. What is object-oriented programming? In *In European Conf. on Object Oriented Programming*, pages 27–6. Springer-Verlag, 1991.

[20] M. J. Swain and D. H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[21] D. Trefethen, Lloyd N. Bau III. *Numerical linear algebra*. Philadelphia: Society for Industrial and Applied Mathematics, San Francisco, CA, USA, 1997.

[22] G. Willems, T. Tuytelaars, and L. Gool. An efficient dense and scale-invariant spatio-temporal interest point detector. *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, pages 650–663, 2008.

[23] G. Willems, T. Tuytelaars, and L. Van Gool. Spatio-temporal features for robust content-based video copy detection. *MIR '08: Proceeding of the 1st ACM international conference on Multimedia information retrieval*, pages 283–290, 2008.