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

IMPLEMENTACIÓN DE UN SISTEMA RECONOCEDOR DE EVENTOS EN VIDEOS, CON UN CLASIFICADOR K-NN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

RENATO VINCENZO ONOFRI SOTO

PROFESOR GUÍA:
BENJAMÍN BUSTOS CÁRDENAS

MIEMBROS DE LA COMISIÓN:
JOSÉ PIQUER GARDNER
JAIME SÁNCHEZ ILABACA

SANTIAGO DE CHILE
MARZO 2014

Resumen

Hoy en día el fácil acceso a la tecnología permite al ser humano registrar, con un mínimo esfuerzo, eventos de interés en su vida. Como consecuencia se genera una gran cantidad de información multimedia, en particular videos, cuyo análisis de contenido es muy difícil de automatizar, siendo deseable el uso de técnicas de minería de datos y visión computacional para aprovechar esta oportunidad. En este contexto, surge la inquietud de clasificar dichos objetos en base a los eventos presentes en ellos, y de esa forma generar una herramienta predictiva que pueda ser usada posteriormente en aplicaciones de diversas áreas, como por ejemplo, en la publicidad.

El presente trabajo de título da cuenta de la implementación de un sistema reconocedor de eventos en video, además de la experimentación con el mismo, la posterior modificación de su componente de clasificación, y la comparación de ambas versiones en términos de eficacia. El tipo de datos que emplea el sistema corresponde a videos de consumidor, los que fueron recolectados por una comunidad científica y agrupados en un dataset de uso público. El sistema se basa en un reconocedor de eventos planteado en un artículo, y está formado por descriptores de características, un módulo de clasificación SVM y un módulo de creación de histogramas. La modificación planteada consiste en cambiar SVM por un clasificador K-NN.

Para cumplir con los objetivos mencionados anteriormente, se sigue la implementación propuesta en el artículo, esto significa que, primero se descarga el dataset y se implementan los descriptores escogidos, posteriormente, se implementa el clasificador SVM y se compara el sistema preliminar con las mediciones de eficacia del artículo, se repite el proceso hasta obtener valores similares y considerar que el sistema ha sido ajustado correctamente. Finalmente, se implementa el módulo K-NN y se comparan ambos sistemas en base a las métricas de rendimiento.

A partir de los resultados de eficacia de las dos versiones, se muestra que el clasificador SVM es una mejor alternativa que K-NN para enfrentar el problema de reconocimiento de eventos en videos de consumidor. Esto es válido para los descriptores con los que se probó el sistema, pero puede no ser cierto si se utiliza otro conjunto de descriptores. Además, se deja en evidencia la dificultad que presenta el manejo de grandes volúmenes de información, y la necesidad de soluciones para su procesamiento.

"Within the swirling mists of time
It's hard to keep a track of year and place"

Agradecimientos

Agradezco a mis padres por haberme entregado la posibilidad de recibir una buena educación, por haber trabajado de sol a sol durante 24 años sacrificando sus propias oportunidades para dármelas a mí. Espero algún día estar a la altura cuando me toque hacer lo mismo por mis hijos.

Agradezco a los profesores que me han hecho clases, de todos he aprendido algo, unos fueron más cercanos y sus enseñanzas tienen que ver con experiencias para la vida, otros me han entregado habilidades como profesional.

Agradezco a mis amigos de siempre, por confiar en mí y por estar presentes en los momentos importantes. El tiempo no los borrará de mi memoria, tampoco lo hará de esta memoria de título.

Tabla de contenido

1. Introducción **1**
- 1.1. Objetivos . . . 3
- 1.2. Metodología . . . 3
- 1.3. Capítulos . . . 4

2. Marco teórico **5**
- 2.1. Bag-of-Words . . . 5
- 2.2. Descriptores . . . 5
  - 2.2.1. SIFT . . . 6
  - 2.2.2. SURF . . . 6
  - 2.2.3. STIP . . . 7
  - 2.2.4. DIFT y DURF . . . 7
  - 2.2.5. MFCC . . . 8
  - 2.2.6. SSIM . . . 8
  - 2.2.7. GIST . . . 8
  - 2.2.8. Color Moments (CM) . . . 8
  - 2.2.9. LBP . . . 9
- 2.3. Vocabulario visual . . . 9
  - 2.3.1. Clusterización . . . 9
  - 2.3.2. Histogramas de frecuencia . . . 10
- 2.4. Clasificadores . . . 10
  - 2.4.1. SVM . . . 10
  - 2.4.2. k-NN . . . 11
- 2.5. Fusión de datos . . . 11
  - 2.5.1. Late Average Fusion . . . 11
- 2.6. Dataset . . . 12
  - 2.6.1. Consumer video . . . 12
- 2.7. Métricas . . . 13
  - 2.7.1. Matriz de confusión . . . 13
  - 2.7.2. Accuracy . . . 13
  - 2.7.3. Precisión . . . 14
  - 2.7.4. Mean Average Precision (mAP) . . . 14

3. Construcción del sistema **15**
- 3.1. Diseño del sistema . . . 15

3.1.1. Reconocedor de eventos en video . . . 15
3.1.2. Dataset . . . 17
3.1.3. Experimentación . . . 20
3.2. Recopilación de datos . . . 20
3.2.1. Descargar dataset CCV . . . 20
3.2.2. Descarga de videos faltantes . . . 21
3.3. Implementación . . . 22
3.3.1. Tabla resumen . . . 22
3.3.2. Iteración Sobre Videos . . . 24
3.3.3. Descriptores . . . 25
3.3.4. Módulo de clasificación SVM . . . 37
3.3.5. Módulo de clasificación K-NN . . . 44
3.3.6. Tabla de parámetros . . . 45

4. Discusiones y experimentación **48**
4.1. Dificultades en la implementación del sistema . . . 48
4.1.1. Creación de vocabularios . . . 48
4.1.2. Costo de extracción de descriptores (problemas con SSD) . . . 50
4.1.3. Interpretación del problema de clasificación . . . 51
4.2. Resultados de la experimentación . . . 53
4.2.1. Evaluación del sistema Baseline . . . 53
4.2.2. Evaluación del sistema Modificado y comparación con Baseline . . . 56

5. Conclusiones **60**

Apéndices **63**

A. Script de descarga de videos **64**

B. Descriptor CM **66**

C. Iterador de videos y extracción de SURF **70**

D. Métricas para Baseline **78**

Bibliografía **80**

Índice de tablas

2.1. Matriz de confusión para dos clases. 13

3.1. Tabla resumen. 22
3.2. Tabla de parámetros. 45

4.1. Comparación según mAP. 53
4.2. Correlación entre métricas. 55

Índice de ilustraciones

2.1. Diferencias entre SURF y DURF 7

3.1. Arquitectura del sistema. 16
3.2. Histograma de videos por clase. 17
3.3. Algunos ejemplos de escenas contenidas en los videos de CCV. 19
3.4. Tipos de división espacial para la extracción de DURF. 32

4.1. Clase 21 y clases compuestas. 52
4.2. Gráficos correspondientes a Overall Accuracy para ambos sistemas 57
4.3. Gráficos correspondientes a Average Accuracy para ambos sistemas 58
4.4. Gráficos correspondientes a Average Precision para ambos sistemas 59

Capítulo 1

Introducción

Día a día la cantidad de información multimedia presente en la Web crece, en particular, en lo que respecta a contenido audiovisual. Debido a la existencia de sitios que permiten a un usuario navegar entre videos, además de subir los propios, es que se hace interesante la idea de poder emplear estas plataformas para construir aplicaciones basadas en el análisis de este contenido, y que permitan una mejora en la experiencia del usuario, además de generar datos que beneficien a diversas comunidades de científicos.

Un conocido ejemplo de investigación realizada sobre videos, es la que se lleva a cabo por medio de TRECVID [1], una serie de conferencias que, a través de competencias, promueven el desarrollo de nuevos mecanismos de búsqueda por contenido en videos. Dichas competencias cubren areas de interés científico que nacen de alguna posible aplicación real, por citar algunos casos: detectar si un video se encuentra o es copia de otro (ya sea en su versión original, o con alguna transformación aplicada), reconocer en qué videos y frames de ellos aparece cierto objeto de consulta (un logo publicitario, un vehículo, herramientas, cubiertos, etc.), detectar comportamiento sospechoso en cintas de vigilancia (alguien corriendo, alguien con un arma, etc.), entre otros. Cada instancia posee un esquema complejo, con un set de datos que serán indexados, un problema computacional asociado, datos de entrenamiento disponibles, pertenecientes a versiones anteriores, y una necesidad a cubrir en caso de que la brecha o gap semántico entre el modelo de computación y ella disminuyan.

Con lo anterior se puede definir el problema en el cual se enmarca esta memoria: clasificar videos en internet según el evento presente en ellos. Esto nace de la necesidad de generar sistemas automáticos capaces de reconocer de qué trata un video (una boda, un partido de fútbol, una presentación musical, un funeral, etc.), y con este tipo de sistema crear aplicaciones que puedan sugerir videos a un usuario de acuerdo a sus preferencias, así como realizar búsquedas evitando el uso de metadatos, lo que permitirían el etiquetado automático al momento de

[1] `http://trecvid.nist.gov/`

subir un video, e incluso, publicidad inteligente de acuerdo a los eventos presentes en un video que se esté reproduciendo. Para esto, se cuenta con la arquitectura de un framework y los algoritmos necesarios para implementarla, basado en *SUPER: Towards Real-time Event Recognition in Internet Videos* [Jiang, 2012], el presente trabajo de título trata sobre la implementación del sistema propuesto y modificación del módulo de clasificación de videos, en conjunto con un análisis comparativo apoyado en los tests originales realizados sobre SUPER y concluir cómo afecta un clasificador K-NN a la eficacia del sistema original.

El interés que genera el tema planteado es debido a que el área de procesamiento de imágenes, orientada al análisis de contenido semántico, es relativamente nueva, y crece el número de aplicaciones que se pueden valer de ella, pues la tecnología de captura audiovisual está al alcance de la mayoría y por tanto, la comunidad científica se encarga de proponer nuevos desafíos técnicos, ya sea como una competencia, o como tema de estudio, con el fin de alcanzar las expectativas del mundo real. En este contexto, en la conferencia ACM Multimedia de 2012, fue publicado *A Fast Video Event Recognition System and Its Application to Video Search* Jiang et al. [2012], que consiste en un buscador de videos según eventos, donde SUPER es uno de sus componentes y fue construído por los mismos autores. Además de esto, el framework se encuentra presente en nuevos proyectos relacionados con detección de escenas violentas en una película [2]. Por lo anterior, resulta un desafío implementar el sistema descrito en Jiang [2012], ya que éste combina técnicas y algoritmos de uso actual por equipos que compiten en las diversas instancias de TRECVID, así como por el hecho de su participación en la conferencia ACM Mutimedia 2012.

En cuanto al desempeño del sistema, éste se obtiene a través de las métricas empleadas por los autores, y que corresponden a las utilizadas en TRECVID, además otras métricas introducidas durante el desarrollo del sistema. Estas métricas permiten realizar comparaciones directas entre sistemas de clasificación. Por otra parte, los datos de prueba originales forman parte de Columbia Consumer Video (CCV) Database [Jiang et al., 2011], base de datos pública que cuenta con 9.317 videos de YouTube, divididos en un set de testing y otro de entrenamiento. El sistema está compuesto por un módulo de extracción de características que permite una cantidad variable de descriptores, ya sean visuales o de audio. Una vez obtenida la información, se utilizan representaciones de tipo Bag-of-Words para cada descriptor y se clasifican, en espacios separados, empleando SVM (Support Vector Machine). Finalmente, se ponderan los resultados individuales de cada clasificación para determinar el evento presente en el video de consulta.

[2] `http://www.yugangjiang.info/research/2012MediaEval/index.html`

1.1. Objetivos

Objetivo general

Implementar un sistema reconocedor de eventos en video, y analizar si se puede mejorar la eficacia del sistema utilizando un clasificador K-NN.

Objetivos específicos

Los objetivos que marcaron el desarrollo de este trabajo fueron:

- Implementar el framework propuesto en el sistema SUPER para reconocimiento de eventos en video.
- Implementar el módulo de clasificación basado en el clasificador K-NN.
- Evaluar el sistema implementado usando la colección de datos CCV Columbia, comparando la eficacia de ambos módulos de clasificación.
- Analizar los resultados obtenidos.

1.2. Metodología

Para lograr los objetivos planteados se utilizó la siguiente metodología:

- Se descargó el dataset de CCV y se creó una lista con los videos faltantes.
- Se implementaron todos los descriptores y se realizaron experimentos para medir sus tiempos de extracción, a partir de los cuales, se dejó fuera del sistema a SSD.
- Se implementó el módulo de clasificación SVM y se entrenó un modelo con los descriptores provistos por los autores. Se siguió un proceso de calibración del sistema a través de la medición y comparación de mAP con respecto al sistema original.
- Se implementó el módulo de clasificación K-NN y se descubrió la necesidad de introducir nuevas métricas.
- Se terminaron de extraer los descriptores y se utilizó Bag-of-Words con SURF para generar sus histogramas. Este proceso requirió la implementación de k-means, y un esquema para gestionar grandes volúmenes de información.
- Se midieron ambos sistemas en términos de eficacia, y se realizó un análisis sobre los resultados.

1.3. Capítulos

Este informe está dividido en 5 capítulos, siendo éste el introductorio, a continuación se explica brevemente el contenido de los demás. En el capítulo 2, correspondiente al marco teórico, se definen conceptos necesarios para entender el presente trabajo. El capítulo 3 trata sobre la construcción del sistema, comenzando por su diseño arquitectural y el tipo de datos a emplear, posteriormente se detalla el proceso de implementación, esto incluye problemas encontrados y las decisiones que se tomaron para solucionarlos. El capítulo 4 presenta los resultados de eficacia del sistema reconocedor de eventos en video, y se realiza un análisis sobre ellos. Finalmente, el capítulo 5 muestra las conclusiones que se pudieron obtener a partir del trabajo de título, y se enumeran posibles extensiones al mismo.

Capítulo 2

Marco teórico

Para entregar mayor claridad sobre los conceptos presentes en esta memoria, ya sean teóricos o técnicos, se presenta este capítulo que cuenta con sus respectivas definiciones y explicaciones breves, agrupadas según categorías, lo que facilita su asociación con el resto del documento.

2.1. Bag-of-Words

Bag-of-Words, propuesto en Sivic and Zisserman [2003], es un framework que se compone de tres módulos, uno de extracción de características, otro para la creación de un vocabulario visual y la asignación de un histograma de frecuencias para cada objeto, y finalmente, la clasificación del espacio de dichos histogramas, permitiendo transformar el problema de decidir a qué clase pertenece un objeto de consulta, a un problema de encontrar la menor distancia entre histogramas. Para esto se requieren datos de entrenamiento, pertenecientes a un dataset, uno o más descriptores para el módulo de características, un método para determinar el vocabulario, y un clasificador para el espacio de los histogramas.

2.2. Descriptores

Son vectores que caracterizan a un objeto multimedia. Toman valores en los reales, en los que cada dimensión recibe un significado según el tipo de característica que se esté midiendo. Permiten enfrentar el problema de la búsqueda por similitud en un espacio métrico, empleando funciones de distancia que representan qué tan parecido es un objeto a otro [Baeza-Yates

and Ribeiro-Neto, 1999]. En este trabajo se considerarán descriptores visuales, que tienen como entrada las imágenes asociadas a los frames de cada video, y un único descriptor de audio, encargado de extraer las características de dicho canal.

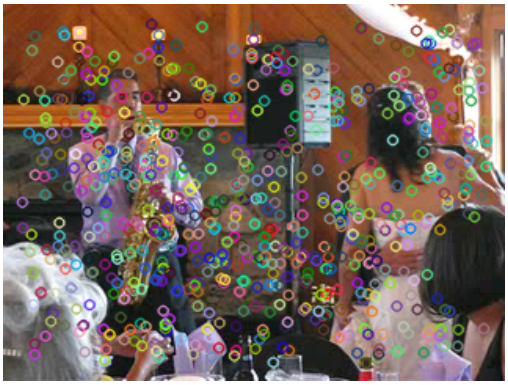
Por otra parte, es conveniente realizar una distinción entre descriptores globales y locales, ambos presentes en el sistema. El primer tipo de descriptor implica un resultado que considera información del objeto multimedia como un todo, mientras que el segundo permite analizar con mayor detalle fragmentos contenidos en él, por lo que su extracción significa recibir como output un conjunto de vectores, en vez de uno solo. Esto es importante desde el punto de vista de la clasificación, ya que para efectos de esta memoria de título, los descriptores globales son entregados directamente al clasificador, sin necesidad de realizar la etapa de creación de histogramas de Bag-of-Words, mientras que los descriptores locales sí son procesados de esa manera.

2.2.1. SIFT

Scale Invariant Feature Transform (SIFT) [Lowe, 2004], se basan en la detección de puntos de interés, que corresponden a fragmentos de la imagen (parches) de distintas escalas, encontrados en base a diferencias de gaussianas (DoG). Esto significa que aplicando filtros gaussianos sucesivamente al parche, y calculando la resta de intensidades de gris entre una iteración y la anterior, se manifiestan los puntos y se almacena la relación entre la iteración (en proporción directa con el sigma del filtro) y el tamaño del punto de interés. Como alternativa a esto se utiliza un método basado en el hessiano de las imágenes [Bay et al., 2008], que permite aproximar las diferencias de gaussianas y disminuye el tiempo de cálculo, perdiendo cierta precisión. Finalmente se calculan histogramas de orientaciones de gradiente en subregiones de la imagen, y por cada punto de interés detectado previamente, se obtiene un descriptor de 128 dimensiones.

2.2.2. SURF

Similar a SIFT, pero en vez de histogramas de orientaciones de gradientes, calcula los descriptores finales, Speeded Up Robust Features (SURF) [Bay et al., 2008], como la respuesta de los parches de los puntos de interés de la imagen a características Haar, que corresponden a sumatorias de intensidades dentro de una ventana deslizante con ponderadores 1 y -1 dispuestos en dicha ventana siguiendo la forma de la característica a la cual son sensibles. La detección de puntos de interés se realiza con el método del hessiano.



(a) Detección SURF de puntos de interés.

(b) Detección densa de puntos de interés.

Figura 2.1: Diferencias entre SURF y DURF

2.2.3. STIP

Spatial-Temporal Interest Points (STIP) [Laptev, 2005], capturan un volumen en el que la variación de intensidades de gris de los pixeles sea tanto en el dominio del tiempo como del espacio bidimensional de la imagen. Para detectar los STIP se emplea el algoritmo de Laptev, que se base en histogramas de orientaciones de gradientes (HOG) e histogramas de flujo óptico (HOF). En ambos casos los histogramas se definen de 72 dimensiones, y el STIP final para ese volumen pasa a ser de 144 dimensiones por la concatenación de HOG y HOF.

2.2.4. DIFT y DURF

Aparecieron por primera vez en Uijlings et al. [2010] y corresponden a aproximaciones de SIFT y SURF utilizando sampleo denso (se evita calcular puntos de interés con métodos como DoG y hessiano, y simplemente se divide la imagen en parches uniformes que se considerarán como si fueran puntos de interés). Posteriormente se repite el proceso de SIFT y SURF. La figura 2.1 muestra las diferencias entre la detección densa de puntos de interés y los algoritmos tales como DoG y hessiano. Los círculos de colores dentro de la imagen indican un punto de interés, y el radio representa la escala en la que fue detectado. Como se puede apreciar, la detección densa elige puntos con la misma separación, mientras que los otros métodos se basan en técnicas más complicadas para retornar zonas con alta variación.

2.2.5. MFCC

Mel Frequency Cepstral Coefficients (MFCC) [Xu et al., 2004], es el único descriptor de audio dentro de esta lista. Se basa en estudios que aproximan la percepción auditiva humana bajo la escala de Mel, que consiste en una representación logarítimica de la señal de origen. Su cálculo involucra la transformada de Fourier, el paso del resultado a la escala de Mel, y utilizar transformada discreta de coseno para finalmente, retornar las amplitudes obtenidas que corresponden a los MFCC.

2.2.6. SSIM

Self-Similarities (SSIM) [Shechtman and Irani, 2007], se obtiene al dividir la imagen en parches mediante sampleo denso, específicamente de 5 x 5 de tamaño. A cada uno de los parches mencionados, se les calcula la correlación con respecto a una ventana circular de 40 píxeles de radio, perteneciente a la misma imagen. Finalmente, los valores de correlación se agrupan en un histograma radial, cuyos bins se determinan particionando radio y ángulo alrededor del parche. Cada uno de dichos histogramas constituye un descriptor local SSIM.

2.2.7. GIST

Este tipo de descriptor global, propuesto en Oliva and Torralba [2001], se basa en filtros de Gabor de 8 orientaciones, con 4 escalas diferentes de la misma imagen. Además, dicho proceso se repite en las subzonas de una malla de 4 x 4 sobre la imagen. Esto da como resultado un vector de 512 dimensiones, por cada frame de video. Cabe destacar que existen versiones que analizan los tres canales de RGB, sin embargo, para efectos de este trabajo GIST se considerará en escala de grises, ya que, de incluir más dimensiones, el vector resultante sería 3 veces más largo.

2.2.8. Color Moments (CM)

Este descriptor global, propuesto en Stricker and Orengo [1995], se obtiene convirtiendo la imagen de entrada al espacio de color L*a*b. Posteriormente, se separa en sus tres canales de color, generando 3 imágenes en escala de grises, las que se particionan mediante una malla de 5 x 5. A continuación, a cada zona de cada malla se le calculan los tres primeros momentos de la distribución de gris en sus píxeles, estos son: promedio, desviación estándar y asimetría estadística (skewness). Finalmente, cada momento pasa a una dimensión en el descriptor. Lo

anterior implica un vector de largo 225 (3 imágenes, 25 subzonas por imagen y 3 momentos por subzona).

2.2.9. LBP

Local Binary Patterns (LBP) [Ojala et al., 2002], define patrones de formas y mide su ocurrencia en la imagen. Para calcularlo se utiliza una ventana de 3 x 3 deslizante, con la que se etiqueta cada pixel con un valor basado en los 8 vecinos definidos por la ventana. Estos valores corresponden a un código binario que representa un tipo de borde, orientación o patrón. Una vez etiquetados todos los píxeles, se obtiene LBP como el histograma de 256 dimensiones resultante de cuantizar las 2^8 combinaciones posibles de códigos binarios.

2.3. Vocabulario visual

Dado un conjunto de datos de entrenamiento, su vocabulario visual para un descriptor en particular corresponde a un subconjunto de todas las instancias de ese descriptor presentes en la colección. Es decir, se realiza el cálculo de descriptores para cada objeto multimedia, y se añaden al conjunto total, luego, se decide un tamaño del vocabulario, y se elige esa cantidad de vectores. Lo anterior se realiza mediante un algoritmo de clusterización. Dicho vocabulario visual representa características que se repiten, o que son muy similares entre todos los objetos [Sivic and Zisserman, 2003].

2.3.1. Clusterización

Consiste en la tarea de encontrar vectores representativos (centroides) de zonas de acumulación de puntos en el espacio de características. En el marco de esta memoria se utilizará el algoritmo de k-means [MacQueen, 1967], que recibe como parámetro la cantidad k de centroides que se deben determinar y aleatoriamente fija k vectores, para luego, iterativamente ajustar la solución hasta cumplir un criterio de término, como es el caso de alcanzar un cierto número de ciclos, o bien, que los centroides no varíen significativamente en dos pasos sucesivos.

2.3.2. Histogramas de frecuencia

Con el vocabulario visual calculado, se toma una instancia del descriptor presente en el objeto, y se busca su vecino más cercano en el conjunto de centroides. Una vez determinado, se aumenta en 1 la frecuencia de ese centroide en el objeto. Se repite el procedimiento hasta terminar con las instancias del descriptor de características, y los resultados se acumulan en el histograma de frecuencias, que en etapas sucesivas será clasificado [Sivic and Zisserman, 2003]. Lo anterior permite reducir la información contenida en el conjunto de un tipo de descriptor local, a un único vector por objeto multimedia.

2.4. Clasificadores

Pertenecen al aprendizaje de máquinas, y se encargan de determinar a qué subconjunto corresponde un nuevo objeto de consulta, basándose en un set de datos de entrenamiento, del que se conocen las clases existentes, y al cuál está asociado cada uno de sus objetos. A continuación se describen los dos tipos de clasificador que se emplearán.

2.4.1. SVM

Un clasificador SVM (Support Vector Machine) [Tan et al., 2005], divide al espacio de vectores de características mediante un hiperplano, es decir, durante el proceso de entrenamiento se encarga de determinar parámetros adecuados que permitan separar los datos en dos sub espacios, de tal forma que, al ingresar una consulta se calcula hacia qué lado del hiperplano le corresponde ser asignada, y por tanto, su clase es aquella que está asociada a esa zona. Esto da cuenta de un clasificador binario (únicamente admite dos tipos de clases), sin embargo, para extender el algoritmo de SVM a un problema como el de este trabajo, se emplea más de un modelo SVM, lo que entrega la posibilidad de realizar múltiples divisiones del espacio y filtrar la zona correcta para una consulta.

Es posible que los datos no sigan una distribución separable mediante estas técnicas lineales, es por esto que se utilizan kernels, los que transforman los datos mediante una función elegida por el usuario, y la división lineal se realiza en el nuevo espacio. Intuitivamente, esta metodología permite realizar separaciones no necesariamente lineales. Para efectos de esta memoria, se trabajará con el kernel de chi cuadrado, cuya función asociada es:

$$\kappa(x, y) = \mathrm{e}^{-p\mathrm{d}_{\chi^2}(x,y)},$$

Donde p es un parámetro a elección del usuario, mientras que $\mathrm{d}_{\chi^{2}(x,y)}$ es la distancia chi cuadrado entre los vectores x e y en $\mathcal{R}^{D}$, cuya definición es:

$$\mathrm{d}_{\chi^{2}(x,y)} = \frac{\sum_{\mathrm{i}=1}^{D} \frac{(x_{\mathrm{i}}-y_{\mathrm{i}})^{2}}{(x_{\mathrm{i}}+y_{\mathrm{i}})}}{2},$$

Cabe destacar que SVM genera modelos que pueden ser almacenados para su posterior uso, sin requerir entrenar nuevamente el sistema, esto es una ventaja desde el punto de vista de la eficiencia del sistema.

2.4.2. k-NN

Un clasificador k-NN [Tan et al., 2005], a diferencia de SVM, no posee una etapa de entrenamiento, y por tanto, su modelo corresponde al conjunto completo de vectores de características. Para resolver una consulta se debe contar con un tipo de distancia entre vectores, como es el caso de la norma euclidiana. A continuación, se realiza el cálculo de distancias de la consulta con respecto a todos los elementos del conjunto, y se eligen los k vecinos más cercanos (previamente se debe haber fijado un valor para k). En base a este subconjunto se obtienen posibles clases para la consulta, ya que cada uno posee su propia etiqueta, y es retornada aquella que aparece más veces.

2.5. Fusión de datos

Es la etapa en la que se debe calcular la ponderación de los distintos descriptores para un objeto, y unir esta información en un solo resultado que permita determinar la clase a la que pertenece [Jiang, 2012].

2.5.1. Late Average Fusion

En este trabajo se decidió entrenar clasificadores independientes para cada tipo de descriptor, por tanto, es necesario determinar una metodología para unir sus resultados, y que

de esa manera, cada uno aporte en la decisión final sobre la clase a la que pertenece una consulta. En este contexto, Late Average Fusion [Jiang, 2012] asigna a cada descriptor el mismo peso, y se obtiene un puntaje promedio ponderado mediante el que se elige la etiqueta correcta. De lo anterior se desprende que es posible que un descriptor indique una clase como respuesta, mientras que los demás decidan sobre otra, por lo que el resultado en conjunto, no necesariamente es el mismo que para cada clasificador por sí solo. Idealmente, esta metodología permite corregir errores del trabajo individual de un descriptor, mediante la adición de más características. Por otra parte, el uso de los mismos ponderadores se debe a que no se conoce a priori el efecto que tiene cada descriptor sobre el dominio de los datos, y no es fácil determinar cuál debería recibir mayor peso. Además de esto, se reduce el riesgo de overfitting, y se obtiene un algoritmo menos complicado de implementar.

2.6. Dataset

Consiste en un conjunto de objetos multimedia que comparten características, tales como: el tipo de contenido, el formato en el que se encuentran, etc. Se emplean para medir el desempeño en sistemas de recuperación de la información, además de permitir la comparación entre dos o más de ellos, pues la colección, idealmente está diseñada para cubrir todos los aspectos relevantes con respecto al problema para el que se creó, por tanto, se considera un terreno para medición objetiva de los sistemas. El diseño de un dataset implica un alto nivel de estudio, recolección apropiada de objetos para ser añadidos, consenso entre académicos, etc. Además de esto, un dataset contiene un conjunto de objetos destinados al entrenamiento de los sistemas, así como uno para realizar testing y obtener métricas del rendimiento. En el contexto de esta memoria, se trabajará con videos de consumidor (consumer video), concepto que se define a continuación.

2.6.1. Consumer video

Los videos de consumidores reflejan los intereses temáticos de los usuarios que los suben a internet y se caracterizan por tener pocas etiquetas, o metadatos, por lo que, la mayor fuente de información es el contenido audiovisual de éstos. Se ha comprobado que para un humano también pueden ser difíciles de categorizar, de acuerdo a los resultados de un experimento realizado con mTurk al momento de crear el dataset de Columbia Consumer Video [Jiang et al., 2011]. Por lo anterior, resulta un desafío generar herramientas automatizadas para clasificarlos.

2.7. Métricas

Para la etapa de experimentación se requieren maneras de medir el desempeño de los sistemas implementados, específicamente la eficacia, ya que se ha dejado fuera el análisis de eficiencia, donde la primera se entiende como la capacidad de retornar la clase correcta para los objetos de consulta, mientras que la segunda corresponde al tiempo que demora el proceso. Para efectos de esta memoria se considerarán las siguientes métricas, cuyas definiciones fueron tomadas desde Baeza-Yates and Ribeiro-Neto [1999].

2.7.1. Matriz de confusión

Es un tipo de visualización, en forma de matriz, que permite mostrar la cantidad de veces que una consulta fue clasificada con cada posible etiqueta. A continuación se muestra un ejemplo para 2 clases:

Tabla 2.1: Matriz de confusión para dos clases.

Clases	A	B
A	60	10
B	5	100

Con respecto a la clase A, el valor 60 corresponde a los verdaderos positivos (TP), 10 a los falsos negativos (FN), 5 a falsos positivos (FP) y 100 a los verdaderos negativos (TN). Esto depende de la clase que se tome como referencia, y puede ser extendido a un mayor número de clases. Cabe destacar que esta matriz da origen a otras métricas, basadas en la agregación de los valores contenidos en ella.

2.7.2. Accuracy

Se define como $\frac{TP}{(TP+FN)}$

En el ejemplo anterior, Accuracy para la clase A corresponde a $\frac{60}{70}$, y para la clase B, $\frac{100}{105}$

En base a esta métrica es posible definir otras dos que serán utilizadas en la evaluación del sistema.

Average Accuracy: Es el promedio de Accuracy para todas las clases.

Overall Accuracy: Es el cuociente entre la suma de los valores de la diagonal y la suma de todos los valores, es decir, corresponde al total de aciertos, dividido por el total de consultas.

Para la matriz anterior, Overall Accuracy es $\frac{160}{175}$

2.7.3. Precisión

Se considerarán dos definiciones posibles para esta métrica, la primera, en el contexto de information retrieval, y la segunda, en términos de la matriz de confusión.

Information Retrieval: Dado un sistema sobre el que se realiza un análisis de eficacia, y que además, es capaz de ordenar sus resultados con respecto a una consulta, en un ranking, se define precisión para cada posición del ranking, como el total de objetos relevantes presentes hasta ese punto, dividido por la posición.

Matriz de confusión: Se obtiene como $\frac{TP}{(TP+FP)}$.

A partir de lo anterior, es posible definir **Average Precision (AP)**, que en el primer caso, corresponde al promedio de las precisiones medidas únicamente en las posiciones en las que se encuentra un objeto relevante. En el caso de la segunda, es el promedio de precisión para cada clase.

2.7.4. Mean Average Precision (mAP)

Esta métrica es ampliamente utilizada en Jiang [2012], y está asociada al contexto de Information Retrieval. Corresponde al promedio de AP para diferentes consultas, es decir, cada consulta al sistema debe generar un ranking, y a partir de él obtener un valor de AP. Por la razón anterior, no es posible calcular mAP en base a una matriz de confusión, ya que no existen más consultas y AP es único.

Capítulo 3

Construcción del sistema

3.1. Diseño del sistema

En esta sección se especifican aspectos arquitecturales y funcionales del sistema. Se detallan los componentes implementados, la base de datos empleada y la forma en la que se experimentó.

3.1.1. Reconocedor de eventos en video

Como se mencionó con anterioridad, un reconocedor de eventos, es un sistema capaz de recibir una entrada, en este caso en video, y retornar una etiqueta que corresponde al evento presente en ese video. Para solucionar dicho problema se plantea el framework Bag-of-Words debido a su utilización en TRECVID, con el más alto rendimiento
[Uijlings et al., 2010]. Dado esto, a continuación se detalla cada módulo del sistema, ya que su arquitectura general se hereda del framework, y se especifica qué es lo que se implementó en el contexto de esta memoria. En la figura 3.1 se puede ver la arquitectura del sistema.

En este trabajo se definieron dos sistemas que fueron comparados, el primero llamado Baseline, que corresponde a una de las configuraciones presentes en el artículo Jiang [2012], y el segundo denominado Modificado, que difiere del anterior en su módulo de clasificación, ya que en vez de SVM se emplea un clasificador K-NN.

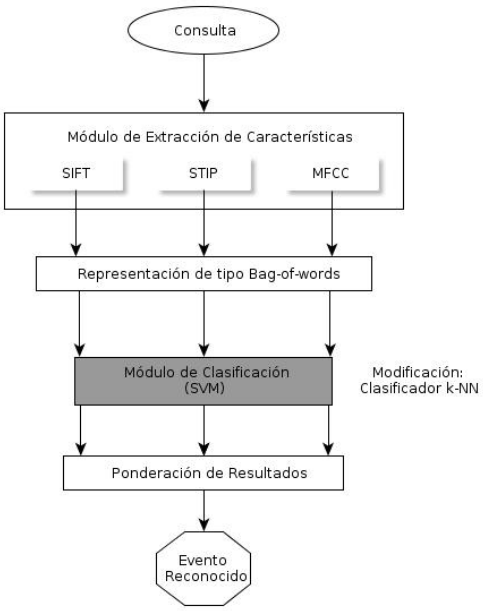


Figura 3.1: Arquitectura del sistema.

Módulo de extracción de características

- Baseline: Se compone de SIFT, STIP, MFCC, DURF, SSIM, CM, GIST y LBP.
- Modificado: Posee exactamente los descriptores mencionados en Baseline.

Módulo de vocabulario visual

- Baseline: Se emplea el algoritmo de k-means por cada espacio de características, definiendo diferentes tamaños de vocabulario según el descriptor correspondiente y basado en los parámetros de Jiang [2012]
- Modificado: Nuevamente se respeta la misma implementación de Baseline y, en ambos casos, se calculan tantos histogramas de frecuencia como la cantidad de descriptores locales, donde las dimensiones de ellos varían según el número de centroides obtenidos por k-means en el espacio de características respectivo.

Módulo de clasificación

- Baseline: Se utiliza un clasificador SVM de kernel χ^2 ya que según Jiang [2012] obtuvo los mejores resultados de eficacia.
- Modificado: A diferencia de Baseline se emplea uno de tipo k-NN.

Fusión de datos

- Baseline: La estrategia utilizada es Late Average Fusion.
- Modificado: Late Average Fusion, al igual que Baseline.

3.1.2. Dataset

La base de datos sobre la que se probó el sistema es Columbia Consumer Video (CCV) [Jiang et al., 2011]. Este set contiene 9.317 videos de Youtube de tipo consumidor, lo que convierte la tarea de clasificarlos en un desafío y, por consiguiente, despierta interés en un sistema automatizado que lo haga. Cada video posee en promedio 80 segundos de duración, con un total de 210 horas para la base de datos completa. En CCV se encuentran presentes 20 clases, las cuales son: Basketball, Baseball,
Soccer, IceSkating, Skiing, Swimming, Biking, Cat, Dog, Bird, Graduation, Birthday, WeddingReception, WeddingCeremony, WeddingDance, MusicPerformance,
NonMusicPerformance, Parade, Beach y Playground. En estricto rigor, no corresponden a eventos solamente, ya que algunas de ellas como Cat o Dog, se denominan objetos, por otra parte, Beach o Playground constituyen escenas. Sin embargo, para efectos del sistema a implementar, esto es transparente, y se les considerará como eventos. El conjunto de entrenamiento viene dado por 4.659 videos, mientras que el de testing por 4.658. En la figura 3.2 se muestra la distribución de videos por clase.

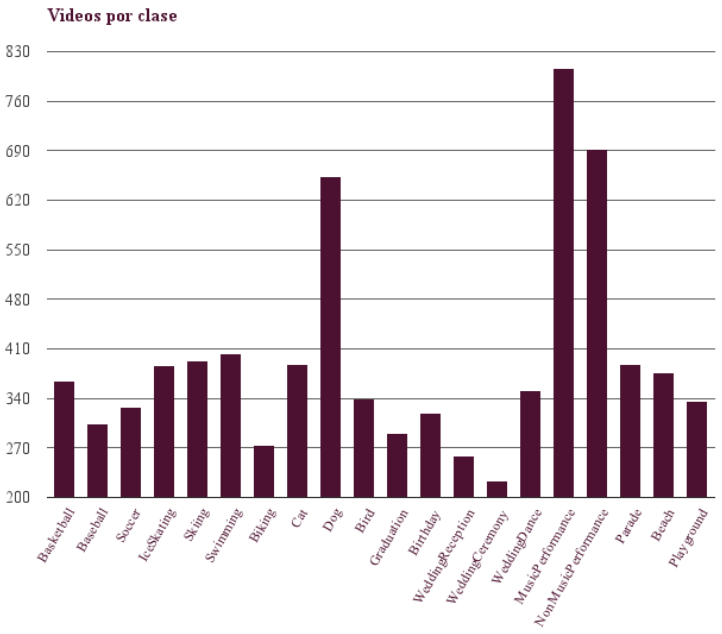


Figura 3.2: Histograma de videos por clase.

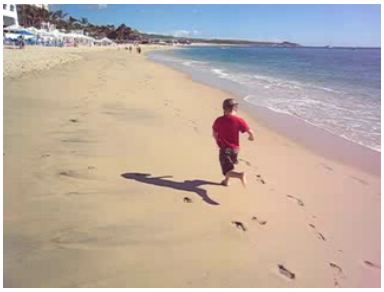
CCV fue creado debido a que los dataset existentes cuentan con videos de entrenamiento en ambientes controlados o su contenido es muy específico, como por ejemplo KTH [Schuldt et al., 2004], que posee acciones humanas fáciles de segmentar con respecto al fondo, o Hollywood Movie database [Laptev et al., 2008] que, a pesar de poseer mayor dificultad de segmentación, no agrega otras clases que podrían ser de interés, ya que se limita, de igual forma, sólo a acciones humanas. Es por esto que al momento de crear CCV se consideraron estadísticas sobre temas de interés de usuarios en internet, y en base a ello, se definieron las clases presentadas anteriormente [Jiang et al., 2011].

El etiquetado de las clases fue hecho manualmente por el grupo de académicos a cargo, mediante un consenso en lo que respecta al contenido de cada video. Esto se debe a la naturaleza de los videos de consumidores, ya que algunas clases poseen una intersección con respecto a contenido que las determina, como por ejemplo,
MusicPerformance y WeddingDance, pues en una boda suele haber música en vivo, o si en un video aparecen un perro y un gato, se puede concluir que se trata tanto de Dog como de Cat.

Cabe destacar que es por la dificultad de clasificación que se probó un etiquetado humano a gran escala mediante la ayuda de una plataforma web, para comparar el desempeño de una persona contra el automático logrado por el sistema Baseline descrito en la sección anterior. Se obtuvo como resultado que a bajos niveles de recall, la precision para ambos era muy similar en algunas categorías como Basketball, pero que a medida que la cantidad de objetos recuperados aumentaba, el humano permanecía con una precisión alta mientras que Baseline disminuía drásticamente [Jiang et al., 2011]. En la figura 3.3 se muestran algunos frames extraídos de la colección y sus respectivas clases asociadas.

(a) Basketball.

(b) Baseball.



(c) Beach.

(d) Bird.

(e) Cat.

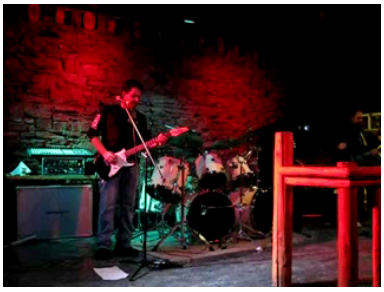
(f) Wedding Ceremony.

(g) Parade.

(h) Non Music Performance.

(i) Ice Skating.

(j) Soccer.



(k) Music Performance.

(l) Swimming.

Figura 3.3: Algunos ejemplos de escenas contenidas en los videos de CCV.

3.1.3. Experimentación

Se clasificó cada video proveniente del set de testing de CCV y con cada consulta se obtuvo un ranking por clase, en él se encuentran todos los videos ordenados según su puntaje de clasificación. A continuación se calcularon valores de mAP, con los que se pudieron comparar, en términos de eficacia, el sistema original con el implementado. Por otra parte, se emplearon métricas basadas en matrices de confusión para comparar a Baseline y Modificado, las que incluyen a Average Accuracy, Overall Accuracy y Average Precision. Para su cálculo se requiere realizar todas las consultas del set de testing, habiendo escogido una combinación de descriptores, y resumiendo su información de etiquetado de clases en una matriz. Posteriormente se opera con las celdas de dicha matriz, y se obtienen las métricas. En lo que respecta al clasificador K-NN, el parámetro K que indica la cantidad de vecinos más cercanos que se consideraron para etiquetar las consultas, también fue objeto de estudio y se utilizaron diferentes valores para él.

3.2. Recopilación de datos

3.2.1. Descargar dataset CCV

Para realizar esta tarea se comenzó por estudiar el sitio web de Columbia Consumer Video dataset [1]. En un principio, se pensó que existían links de descarga directa. A pesar de que se sabía que los videos fueron recopilados desde Youtube, se supuso que los académicos que reunieron CCV, habían conservado una copia de la colección en sus servidores para asegurar su integridad. Lamentablemente, lo anterior no resultó ser cierto, y lo que se obtuvo, en el link de descarga, fue un conjunto de archivos de texto con los id que Youtube asignó a los diferentes videos, de tal manera que para visitarlos se concatenaba el id en la url, además de los nombres de las clases, las etiquetas de clase para cada video, y los descriptores SIFT, STIP y MFCC de los 9317 videos. Dado el escenario descrito anteriormente, fue necesario determinar una metodología para descargar la colección, la cual consistió en dos etapas. En primer lugar, encontrar una manera para descargar videos desde Youtube, y en segundo lugar, automatizar dicho procedimiento.

La solución para la primera parte consistió en la instalación del programa llamado youtube-dl [2]. Dicho programa permite, a través del terminal de linux, descargar videos de múltiples páginas web, en particular, de la que se requería en esta tarea. Para ello se debe ejecutar youtube-dl y entregarle la url deseada, además de parámetros opcionales, que incluyen extracción de audio, modificación del formato del video, entre otros. Se consideró como alternativa

[1] `http://www.ee.columbia.edu/ln/dvmm/CCV/`
[2] `http://rg3.github.io/youtube-dl/`

el uso de plugins para un browser, sin embargo, la manera de ejecución de youtube-dl, entregaba un terreno favorable para la automatización de la descarga masiva de videos, por lo que se experimentó con algunas url, y se comprobó que el software funcionaba bien, de esta forma se avanzó a la siguiente etapa del proceso.

Se escogió Python para crear un script [3] que llamara al programa iterativamente leyendo el archivo con los id, y concatenándolos con la url de Youtube. La elección se debió a la gran cantidad de ejemplos presentes en la web sobre Python, por lo que su implementación fue rápida. En un principio se utilizó una versión simple de script que sólo llamaba al proceso creando un pipe, y se supervisó el output de la terminal. Una vez detectados problemas en la descarga, se conectó la salida de error del proceso a una variable, para escribir en un log todos aquellos videos y sus id que fueron conflictivos. Una vez hecho esto, se dejó el script funcionando por un tiempo total estimado de 4 días, distribuido a lo largo de una semana, admitiendo ciertas pausas, y comenzando desde la última línea leída.

Se descubrió que el dataset no estaba de forma íntegra, pues al revisar los errores en el log, la mayoría se trataba de videos que youtube-dl no pudo encontrar. Verificando manualmente, se corroboró que habían sido eliminados del sitio, y que no era un problema con el software. La razón de videos perdidos de la colección se estimó en 1 de cada 10.

3.2.2. Descarga de videos faltantes

Durante el desarrollo de esta memoria, fue necesario comunicarse con los autores de Jiang [2012] para resolver una duda con respecto a las clases, y una duda asociada con la manera de clasificar descriptores globales. La primera se trataba de que algunos videos no tenían etiqueta de entre las 20 mencionadas en Jiang et al. [2011], por lo que se pensaba que faltaba cierta información, o que existía una clase para representar videos de consumidor que no pertenecían a ninguna, y que se encontraba en CCV para permitir mayor generalizabilidad a los modelos. La segunda se relacionaba con la incertidumbre entre clasificar directamente los descriptores globales, o generar algún tipo de histogramas a partir de ellos y hacerlos parte de Bag-of-Words.

Ante esto, se le envió un correo al profesor Yu-Gang Jiang, manifestándole las inquietudes, y se le mencionó que la descarga de CCV estaba en un 87 % a causa de usuarios que eliminaron o hicieron privados sus videos. Ambas dudas fueron respondidas, indicando que, efectivamente existe una clase de ejemplos negativos, y que los descriptores globales fueron clasificados directamente, sin pasar por Bag-of-Words. Por otra parte, ofreció la posibilidad de enviar los videos faltantes, siempre y cuando se utilizaran en un contexto científico, y con propósitos únicamente de investigación. Se aceptaron los términos, y se realizó un catastro de los videos perdidos, para así entregarle un archivo con los id.

[3] Está disponible en Apéndices, el script de descarga masiva de videos.

Finalmente se completó la colección, sin embargo, se encontraron 10 videos, 5 de entrenamiento y 5 de testing, que no pudieron ser abiertos por OpenCV a causa de errores en su codificación. Se decidió no intentar pedirlos, ya que incluso considerando sus descriptores con valor 0, el impacto que tendrían sobre los resultados no sería significativo. Además, el envío no fue inmediato, y se requirió mantener la comunicación por varias semanas para hacer las gestiones pertinentes.

3.3. Implementación

En esta sección se explica el proceso de implementación del sistema, incluyendo los problemas que surgieron y las decisiones que se tomaron para solucionarlos. A continuación se muestra una tabla que resume dichos contenidos. Posteriormente se detalla el desarrollo de cada componente. Para finalizar, se muestra una tabla con datos técnicos y parámetros de cada componente.

3.3.1. Tabla resumen

Tabla 3.1: Tabla resumen.

Componente	Idea Inicial	Problemas	Implementación Final
SIFT, STIP, MFCC	Utilizar archivos provistos en el sitio de CCV.	Fue necesario modificar el formato para entrenar el clasificador.	Se cumplió con la idea inicial y se utilizó un script para cambiar los formatos.
CM	Utilizar implementaciones existentes.	No se pudo encontrar una implementación en la red.	Se implementó en C++ junto con OpenCV.
LBP	Utilizar implementaciones existentes.	No hubo problemas asociados.	Se utilizó una implementación encontrada en la red (C++ y OpenCV).
GIST	Utilizar implementaciones existentes.	Fue necesario modificar la interfaz para leer imágenes (OpenCV).	Implementación en C++ proveniente de internet, junto con modificaciones hechas en OpenCV.

Componente	Idea Inicial	Problemas	Implementación Final
DURF	Utilizar implementaciones existentes.	No se pudo encontrar una implementación en la red. Se implementó en C++, combinando la funcionalidad de OpenCV, que cuenta con detectores densos de puntos de interés y extracción de SURF. Sin embargo, hubo una baja eficacia en resultados preliminares.	Se modificó la implementación de OpenCV, para utilizar el detector de puntos habitual de SURF, es decir, se calcularon los descriptores SURF en vez de DURF.
SSD	Utilizar implementaciones existentes.	Se estimó que con la implementación encontrada, la extracción de SSD de toda la colección tomaría un tiempo del orden de meses, incluso disminuyendo los tamaños de los frames.	Se eliminó del sistema y no se implementó SSD.
Iteración sobre la colección	Implementarla en C++ junto con OpenCV.	No hubo dificultades.	Se aprovechó la funcionalidad de OpenCV para leer cada frame de los videos de la colección.
Módulo SVM	Utilizar implementaciones existentes.	Se encontraron diversas implementaciones de SVM, pero ninguna cumplía todos los requisitos (tipo de kernel, manejo de más de dos clases, etc.)	Se escogió una de las implementaciones que se encontraba en C++, y se modificó para añadirle las características faltantes. Se optó por utilizar un kernel diferente.
Módulo K-NN	Utilizar implementaciones existentes.	La implementación en C++ que se escogió, no admitía el uso de más de un descriptor al momento de responder las consultas.	Se modificó de gran manera la implementación, manteniendo la original como template, y se añadió la funcionalidad requerida.

Componente	Idea Inicial	Problemas	Implementación Final
Generación de Vocabularios	Utilizar OpenCV y sus clases asociadas al framework Bag-of-Words.	Hubo problemas con el gran volumen de datos (28 GB para SURF), y la implementación no fue capaz de soportarlo. Además, el tiempo de ejecución de cada iteración de k-means superaba las 5 horas.	Se hizo una implementación propia de k-means, y se utilizó una heurística para reducir el volumen de datos a 3 GB.
Generación de Histogramas	Implementarlo en base a scripts sencillos.	No hubo dificultades.	Se implementó en C++, y se escribió en archivos de texto cada resultado.
Métricas	Implementarlas dentro de cada módulo de clasificación y generar outputs a archivos de texto.	No hubo dificultades.	Se implementó en C++ (debido a que los clasificadores se encontraban en este lenguaje), se generaron matrices de confusión como objetos, y se iteró sobre ellas para generar las métricas.

3.3.2. Iteración Sobre Videos

Se programó en C++, en conjunto con OpenCV, un esquema para recorrer la colección de videos [4]. Se escogió esta plataforma debido a que OpenCV entrega múltiples herramientas para procesar videos y los frames contenidos en ellos, además de clases que abstraen los comportamientos de los elementos multimedia mencionados, y permiten acceder a los valores de sus píxeles para ejecutar algoritmos de visión computacional.

El procedimiento para iterar sobre el set de datos comienza con la creación de un archivo .txt sobre el que se escribirá el output del programa, el cual consiste en que cada línea lleva el id del video, y separados por espacios, los valores de cada coordenada del descriptor que se está calculando. A continuación, se emplea una función que carga en un vector de la librería estándar de C++, los path de cada archivo encontrado en el directorio que se ingresó como parámetro, por ejemplo, la ruta a la colección de training. No es un inconveniente si existen archivos que no sean videos, ya que el programa los filtra.

[4]Está disponible en Apéndices, el código de iteración de videos junto con la extracción de SURF.

Una vez completado el paso anterior, se emplean funciones de OpenCV para abrir cada video a partir de su path, el que se recibe como string, por lo que se usa un método de la librería de string de C++ para convertirlo en un puntero a char. Este último parámetro se le entrega al constructor de un objeto de la clase VideoCapture que permite acceder a cada uno de los frames del video. Posteriormente, se itera sobre cada frame eligiendo uno cada dos segundos, es decir, al momento de pedir el siguiente frame a VideoCapture, se le incrementa en dos veces su valor de frames por segundo. Esto se hereda del paper original, y es la resolución máxima que emplearon los autores, además de ser la base para experimentos posteriores, en los que disminuyeron el número de frames por video, por ejemplo, tomando uno cada cuatro segundos. La decisión de conservar la resolución base en vez de una versión con down sampling, se debe a que existe el supuesto de que con mayor cantidad de frames la eficacia del sistema será más alta, esto se apoya en los resultados de Jiang [2012].

Una vez llegado al nivel de frames, cada uno de ellos se representa con un objeto de tipo Mat, que es la clase que entrega OpenCV para trabajar con imágenes, además de poder usarse como contenedor. Es en esta etapa donde el proceso depende del tipo de descriptor elegido. En caso de ser global (LBP, CM o GIST), se entrega el frame y se retorna el valor de su descriptor, el cual se acumula en un vector de C++ que fue inicializado en cero antes de entrar a este ciclo, este vector almacena la suma coordenada a coordenada de los descriptores de cada frame, y su tamaño es el mismo que el de ellos. Posteriormente, cuando todos los frames del video han sido procesados, se divide cada coordenada del vector acumulador por el total de frames, y dicho vector pasa a ser el descriptor promedio. Finalmente, el descriptor promedio se guarda en el archivo de texto de output.

En caso de tratarse de un descriptor local (Self Similarities o DURF), se utilizan dos funciones de escritura en archivos, y en vez de un único archivo de texto, son dos los que deben ser creados al comenzar el programa. Esto se debe a que el primero de ellos es para almacenar por cada línea, el id del video, el número de frame y las coordenadas del descriptor separadas por espacios. Por otro lado, el segundo archivo, complementa estos datos y sus líneas están en correspondencia, en él se guarda id, número de frame y la información de ubicación del descriptor, esto se refiere al cuadrante que ocuparía si la imagen se divide en 2 x 2, o si se divide en 3 x 1. Esta información será ocupada posteriormente para evitar calcular los descriptores más de una vez, y poder generar histogramas de estas subregiones que añadan información al histograma final de 4000 dimensiones.

3.3.3. Descriptores

SIFT, STIP y MFCC

Los histogramas de características para estos tres descriptores se entregan junto con la descarga de CCV, por lo que no tuvieron que ser implementados. Sin embargo, fue nece-

sario modificar los archivos provistos, para llevarlos al formato aceptado por el clasificador PmSVM. Esto se realizó mediante scripts sencillos.

CM

Este descriptor [5] fue implementado en C++, junto con la librería OpenCV. Esto se debe a que la búsqueda de código en la red no entregó resultados confiables, pues existen técnicas basadas en momentos en diversos dominios, como por ejemplo, lo relacionado con los contornos de las figuras presentes en una imagen, y no se encontró una para los tres canales. Además, se requería que el análisis fuera hecho en el espacio de color L*a*b, por lo que OpenCV y una implementación propia permitían mayor personalización. Se crearon dos funciones principales, una para calcular un vector de 9 dimensiones con los CM de los tres canales de L*a*b, y otra para dividir la imagen en N x M zonas. Esta última es necesaria, ya que en el artículo original se concatenan los descriptores CM de 25 subzonas, las que se obtuvieron al particionar los frames con una grilla de 5 x 5. La combinación de las funcionalidades anteriores entrega un vector de 225 dimensiones que será usado en el paso de entrenamiento.

Para calcular un descriptor CM sobre un objeto Mat de OpenCV, se siguen las definiciones de los 3 primeros momentos, donde cada uno es una medida estadística de la distribución de intensidades de los píxeles en dicho canal. Por tanto, el proceso implica separar la imagen original en 3 imágenes, una por canal, e iterar para cada una de ellas a través de sus píxeles, agregando en variables sus valores estadísticos. Algunos momentos dependen del primero, por tanto se requiere volver a iterar una vez que se obtuvo el primero. La función anterior recibe un objeto de tipo Mat, por lo que, para realizar las particiones en 5 x 5, bastó crear una función de OpenCV que recibiera la imagen original, y creara un vector de Mat, con lo que se hace una llamada al método de cálculo de CM y se concatenan los vectores de 9 dimensiones. Para dividir la imagen se emplean funcionalidades de OpenCV que permiten crear un objeto Mat y llenar sus filas y columnas con la submatriz de otro objeto Mat, en este caso, la imagen de consulta. Cabe destacar que no es relevante el orden en que se asignan las coordenadas de los momentos y de los canales al vector final, lo importante es que sea de la misma forma para todas las imágenes sobre las que se va a calcular el descriptor. Esto guarda relación con la distancia Euclidiana que se empleó en las comparaciones de similitud, y podría no seguir siendo válido al elegir otra distancia.

A continuación, se le agregó al esquema iterador de videos, el header correspondiente a CM, y se hicieron pruebas para medir el tiempo que tomaba el cálculo de dicho descriptor en relación a los frames que procesaba por segundo, y un promedio de videos por minuto. El resultado fue favorable, y se obtuvo una estimación de 15 videos por minuto, por lo que en una hora se podían procesar 900 videos, lo que implica un total de 10 horas para la colección completa, lo que se divide en 5 para cada subconjunto, es decir, entrenamiento y testing.

[5] Está disponible en Apéndices, el código para calcular el descriptor CM.

Dado el contexto anterior, se concluyó que CM no requería paralelismo en su cómputo y corroboraba la factibilidad del uso de CM en el sistema a implementar.

SSD

La implementación utilizada para extraer este tipo de características proviene del sitio web [6], y fue programada por los mismos autores de Chatfield et al. [2009], donde se define este descriptor como una variante al propuesto en Shechtman and Irani [2007]. Además, cabe destacar que es precisamente dicha variante la que emplearon los autores del sistema reconocedor de eventos en video. El código está escrito en C++ y posee interfaz para Matlab, también se ofrece la posibilidad de descargar una versión antigua, únicamente en Matlab. Se asegura que la versión actual funciona más rápido y está optimizada. Esto es importante, ya que SSD es un descriptor local, y requiere iteraciones para cada parche de una imagen con respecto a su vecindario.

Se buscaron alternativas a esta implementación, sin embargo, no se encontró ninguna. Sobre este punto, se debe aclarar que a SSD, los autores de SUPER lo llaman SSIM, lo cual, se puede prestar para confusión, pues existe otro descriptor que recibe ese nombre, y se trata de Structural Similarity. Por tanto, si se busca en la red SSIM, como se hizo al inicio de este trabajo, se pueden encontrar diversas implementaciones, incluso algunas que emplean OpenCV, pero todas ellas se refieren a Structural Similarity. Por otra parte, se debe recordar que el código del sitio mencionado previamente, fue hecho por los mismos autores de dicho descriptor, por lo que se decidió que era suficientemente confiable, y que sería la única implementación con la que se trabajaría.

Se comenzó por compilar la versión en C++, y se siguieron las instrucciones provistas en la documentación. Se pudieron apreciar dos funciones importantes, una de ellas para calcular SSD sobre una imagen, y otra para normalizar y filtrar los vectores obtenidos. La primera poseía dos versiones, una con resultados exactos, y otra con aproximaciones para mayor eficiencia. La segunda, se trataba de un paso complementario ideado por los autores para descartar vectores que no cumplían con ciertos criterios [**?**], y para normalizar todos los descriptores. Esto último, es necesario pues las imágenes pueden variar en sus tamaños, y ya que Self Similarity es un tipo de histograma, y por tanto, posee bins, la cantidad de votos depende de dichos tamaños. Debido a lo anterior, los autores recomendaban realizar un paso de normalización, ya sea con el método provisto por ellos, o con una función propia.

Otra observación sobre la implementación, es que no se encontraba en OpenCV, o en otra librería de procesamiento de imágenes, y la interfaz para recibir una imagen consistía en una matriz de tres canales vectorizada, es decir, un puntero de double que almacena los datos en una única dimensión, y del que el programa es capaz de decodificar las separaciones para

[6] `http://www.robots.ox.ac.uk/~vgg/software/SelfSimilarity/`

entenderla como una matriz. Por tanto, para poder utilizar directamente SSD, sin modificar de gran manera su código fuente, se decidió emplear OpenCV para abrir las imágenes y crear una función que las pudiera vectorizar y llevar al formato adecuado para ser procesadas. Además, se decidió aprovechar la funcionalidad provista, y utilizar el método de normalización y filtrado existente. La vectorización fue sencilla de implementar y se siguió el esquema column-major para recorrer la imagen.

A continuación, se hicieron pruebas para calcular SSD de una imagen, y se pudo apreciar que tomaba aproximadamente 30 segundos para una imagen de 320 x 240, mientras que en el caso de imágenes de mayor tamaño, el tiempo llegaba a los 5 minutos. Esto era sin realizar normalización ni filtrado, pues al añadirlos, en el caso de la imagen de 320 x 240, pasaron 30 minutos y el programa aun no terminaba, lo que llevó a descartar inmediatamente este paso de refinamiento, y a pensar en normalizar manualmente con una función propia. Esto último se justifica porque un video de CCV posee en promedio 40 frames, y 40 x 30 minutos es excesivo para procesar un solo video. Se concluyó que la única forma de poder calcular los valores de SSD de la colección era reduciendo los tamaños de los videos a 320 x 240, o incluso menos. Este proceso no aumenta la complejidad ya que OpenCV ofrece funcionalidad optimizada para escalar imágenes.

Se tomó la colección de entrenamiento y se comenzó a procesar para extraer los descriptores, con el objetivo de estimar el tiempo promedio por video, ya que, considerando los datos mencionados previamente se veía infactible la inclusión de SSD en el sistema final. El tiempo promedio, para un video cuyas imágenes fueron escaladas al tamaño 320 x 240, es de 20 minutos. Esto significa que por cada hora se pueden procesar 3 videos, es decir, en 3000 horas (ya que son aproximadamente 9000 videos) se tendría la colección CCV completa. Traduciendo a días, son 125, por lo que se requiere un computador funcionando día y noche por 4 meses para completar los cálculos.

Analizando las posibilidades de paralelizar el problema, se desestimó esta opción, ya que en el computador que se utilizó, se podría haber reducido dicho tiempo a la tercera parte. Sin embargo, aun así se requería gran uso de esta herramienta, y se debía tener en cuenta que habría que pausar en ciertos momentos la ejecución. Por otra parte, no se contaba con los recursos suficientes como para añadir otros computadores, ya que la información era muy pesada como para emplear el almacenamiento de la universidad, y gestionar un mecanismo para integrar todos los sub cómputos sería un trabajo del nivel de una memoria. Por estas razones se desestimó la utilización de SSD en el sistema en construcción.

LBP

La implementación de este descriptor fue descargada desde el sitio [7]. Esta decisión se debe a que dicho código está hecho en C++, junto con OpenCV. A pesar de que existen diversas alternativas en la red, la plataforma sobre la que está programada esta opción fue preferida por su compatibilidad con el resto del sistema en construcción. La descarga provee un archivo de testing que muestra las modalidades soportadas con respecto a LBP. Esto se debe a que dicho descriptor posee variantes que se manifiestan a través de las combinaciones de sus parámetros. En particular, la configuración por defecto de este ejecutable coincide con la utilizada por los autores del sistema original.

Basándose en el archivo de testing mencionado, se creó un programa para extraer LBP y se respetaron los parámetros por defecto. Además, se incluyó un paso de normalización de tal forma que las coordenadas de los vectores sumaran uno. Lo anterior se debe a que LBP es un histograma, por tanto la implementación debe retornar en cada coordenada, el número de veces que esa característica estuvo presente en la imagen de entrada, lo que es directamente proporcional a las dimensiones de la misma. Esto complica la futura comparación de descriptores, ya que no todos los videos tienen los mismos tamaños de frames. Para completar esta etapa bastó con dividir cada coordenada por la suma de todas ellas.

Posteriormente, se agregó la funcionalidad al iterador de videos, y se siguió el mismo esquema que para los demás descriptores globales, sin necesidad de modificaciones en la interfaz, ya que esta implementación recibía matrices de OpenCV directamente. A continuación, los valores de cada descriptor fueron guardados en un archivo de texto, manteniendo el id del video para su futura clasificación. Cabe destacar que se estimó que el proceso de extracción de LBP tenía una velocidad de 20 videos por minuto, lo que se traduce en 4 horas aproximadamente, tanto para la colección de entrenamiento como para la de testing, es decir, un total de 8 horas para CCV.

GIST

La implementación de este descriptor global proviene del sitio [8]. Esto se debe a que dicha implementación está señalada en el paper de SUPER, por tanto, para ser fiel al sistema original, y minimizar diferencias, se descargó y decidió utilizar dicho código. Este descriptor viene bajo el nombre Lear GIST Implementation, y está programado en C. No emplea librerías gráficas como OpenCV, ya que utiliza como interfaz una estructura llamada standalone_image creada por los autores. Dicha estructura representa a una imagen, y posee variaciones que permiten leer imágenes en color, es decir, de tres canales. Internamente, el código es de bajo nivel, y emplea punteros a float para manejar las intensidades de los píxeles.

[7] `https://github.com/nourani/LBP`

[8] `http://lear.inrialpes.fr/software`

Esta implementación admite cargar imágenes, pero exige que posean el formato pgm o ppm, por lo que no fue considerada suficientemente versátil como para añadirla directamente al sistema que se construyó. La solución fue utilizar un paso de preprocesamiento en OpenCV, en un archivo en C++, aprovechando la compatibilidad de C y C++. Para esto se consideró Lear GIST como librería y se llamó a sus funciones, además de agregar un método que cargaba imágenes en cualquier formato, y las iteraba para crear una estructura del tipo standalone que sería el input de las funciones de cálculo de GIST. Esta decisión se debe a que resulta menos complicado de implementar, y no se sacrifica mucho tiempo ya que OpenCV posee un manejo rápido de matrices. La alternativa hubiese sido modificar todo el código de Lear GIST para hacerlo compatible con las estructuras de OpenCV, y así evitar este paso de iteración sobre la imagen de entrada, sin embargo se concluyó que la ganancia en eficiencia no sería importante, e incluso era imperceptible.

Cabe destacar que Lear GIST posee variantes del descriptor, en particular, se distinguen dos tipos, uno de ellos es para imágenes a color, mientras que el otro es un GIST en escala de grises. En el caso de SUPER, los autores emplearon la versión en escala de grises, con 8 orientaciones, 4 escalas y una malla de 4 x 4 sobre la imagen de entrada, por lo que su vector resultante es de 512 dimensiones (8 x 4 x 4 x 4). Si se tratara de los mismos parámetros para el descriptor en colores, las dimensiones aumentarían a 1536, ya que el proceso se realiza para los 3 canales (512 x 3). que como es de esperar, no requiere un proceso adicional de cuantización, como es el caso de bag of words, para descriptores locales Con esta funcionalidad completa para los formatos usuales de imágenes se procedió a calcular un vector de GIST para cada video, que corresponde al promedio de los descriptores obtenidos para cada uno de sus frames. De esta forma, se recurrió al esquema de iteración sobre videos descrito con anterioridad. Las primeras pruebas indicaron que el proceso sería bastante lento por cada frame, lo que aumentaba de gran manera el tiempo total para la colección completa de CCV. Es por esto, que se tomó la decisión de escalar las imágenes con tamaños superiores a 320 x 240, y dejarlas con esos valores máximos para el ancho y alto, esto se debe a que se comprobó que el tiempo de cálculo disminuía en gran medida para imágenes de menores dimensiones.

La posibilidad antes descrita se había analizado para Self Similarities pero se desestimó cuando se eliminó del alcance de esta memoria dicho descriptor local. Sin embargo, en esta ocasión, GIST sí entregaba tiempos razonables para un semestre de trabajo, pero implicaban bastantes horas de cálculo. Además, al haber recibido los videos faltantes por parte de los autores de SUPER y CCV, se descubrió que 320 x 240 eran las dimensiones de sus videos, por lo que se concluyó que ellos hicieron este paso de escalamiento, pues en los videos descargados desde Youtube una gran cantidad de videos superaba ampliamente dicho tamaño. Por lo argumentado anteriormente, se agregó el preproceso de escalar las imágenes que superaban los 320 x 240.

DURF

Este descriptor local fue programado en C++ junto con OpenCV, esto se debe a que no se encontró una implementación completa de DURF en la red. Entre las alternativas analizadas, incluyendo OpenCV propiamente tal, se pudo notar que SURF sí era soportado. Es preciso recordar que DURF, propuesto en Uijlings et al. [2010] es una variación de SURF, en la que la detección de puntos de interés se realiza con dense sampling, en vez del método del Hessiano. Es decir, si se abstrae el comportamiento de ambos, y se separa en detección de keypoints y extracción de características, esta última es idéntica para ambos, mientras que la primera es distinta.

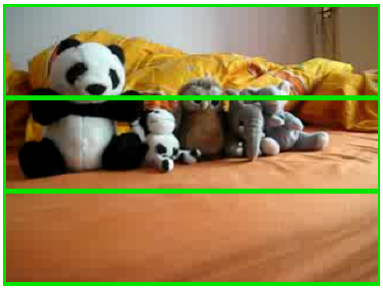
Cabe destacar que DURF no se encontró completamente implementado, pues el artículo Uijlings et al. [2010] fue publicado recientemente, y es en él, donde se analiza esta variante de SURF, obteniendo buenos resultados, además de contemplar su uso para sistemas más rápidos de clasificación de videos. Por tanto, a pesar de que DURF o DIFT (versión densa de SIFT) han podido implementarse sin mayor complicación a partir de la abstracción mencionada previamente, no ha habido mayor demanda para publicar código específico para hacerlo directamente y en un único módulo. Debido a lo anterior, se investigaron implementaciones de SURF que cumplieran con la separación en etapas de su proceso, permitiendo modificar la detección de keypoints, y conservando la extracción de características.

Se pudo notar que OpenCV ofrecía el algoritmo de detección de puntos de interés propio de SURF, cuyo resultado se almacenaba en un vector de Keypoint (clase destinada al manejo de este tipo de datos) y, posteriormente, se debía emplear otra clase para realizar la extracción. Las clases mencionadas son: SurfFeatureDetector y SurfDescriptorExtractor. Por las razones anteriores se decidió utilizar esta librería. En un principio, se pensó que dichas clases eran únicas, por lo que debía implementarse un esquema de detección densa de puntos de interés, y luego transformar los datos a la clase Keypoint para poder aprovechar el objeto SurfDescriptorExtractor. Sin embargo, en la documentación se pudo ver la existencia de otras versiones de detectores de keypoints, que permitían la combinación de algoritmos. En particular, OpenCV provee una clase de dense sampling llamada DenseFeatureDetector. De esta forma, la implementación se simplificó de gran manera, y lo único restante fue verificar los parámetros para el objeto DenseFeatureDetector de forma de respetar el esquema de Uijlings et al. [2010]. Cabe destacar que no se especifica en el paper de SUPER qué valores se le entregaban al detector de keypoints, por lo que se decidió emplear los mismos que en el artículo de Uijlings debido a que los autores del reconocedor de eventos en video declaraban que su esquema de DURF era basado en él.

Con la funcionalidad anterior cubierta, se añadió DURF al programa iterador de videos, y se hicieron cambios para adaptarlo a los requerimientos de un descriptor local. Es necesario recordar que en el caso de los descriptores globales, basta con obtener el vector de características de cada frame y promediarlos para generar uno que represente al video completo. En cambio, en los descriptores locales, usualmente son más de uno por frame, y promediar en estas circunstancias implicaría una mayor pérdida de información, pues se trata de diferentes

(a) Imagen original, o grilla de 1 x 1.

(b) Grilla de 2 x 2, 4 sub imágenes.



(c) Grilla de 3 x 1, 3 sub imágenes.

Figura 3.4: Tipos de división espacial para la extracción de DURF.

zonas de la imagen, y el tamaño del conjunto no es fijo, de manera que, condensar todos esos datos no es recomendable.

Debido a la necesidad de un único vector de características por cada video, se implementó el esquema de Bag-of-Words. De esta forma, se agrupa la información de los descriptores locales en un sólo histograma. En particular, la configuración para este sistema consistió en calcular los descriptores DURF para cada frame, y considerar tres tipos de división espacial a manera de grilla sobre la imagen. La primera es de 1 x 1, es decir, la imagen sin recortar, la segunda es de 2 x 2, con lo que se obtienen 4 subzonas, y la tercera es de 3 x 1, lo que implica 3 adicionales. A cada uno de estos 8 tipos de zona se le creó un vocabulario de tamaño 500, empleando el algoritmo de k-means para clusterizar los descriptores y fijar los 500 centroides. Cabe destacar que, las disposiciones espaciales permiten aumentar la resolución de los histogramas, ya que la imagen completa, idealmente va a conservar los descriptores más relevantes del total, mientras que un cuadrante posee sus propios **relevantes locales**, y con las 500 palabras se pretende corregir una posible falta de vectores menos **dominantes**. La figura 3.4 muestra los tres tipos de división espacial para DURF.

Para lo anterior, nuevamente se consideró la funcionalidad provista por OpenCV, y se utilizó la clase BOWKMeansTrainer, que recibe una matriz con un descriptor por cada fila y entrega en otra matriz el vocabulario según los parámetros mencionados. En este paso, se asigna a la matriz de input el conjunto completo de descriptores, para una de las configuraciones espaciales, es decir, sin distinguir frame o video se agrupan todos. Posteriormente, se ejecuta k-means y se guarda la matriz con el vocabulario en un archivo.

Para efectos de persistir los datos se empleó la clase FileStorage que proporciona la librería, y que permite manejar de forma sencilla archivos .xml o .yml, con un formato específico para guardar las matrices de OpenCV. En particular, se almacenó cada uno de los 8 vocabularios en un archivo distinto. Cabe destacar que los descriptores DURF del set de training, fueron

persistidos usando la misma metodología.A continuación, fue necesario calcular histogramas para cada video, por lo que se creó un programa para cargar los vocabularios así como los descriptores de entrenamiento, que habían sido almacenados en un mismo archivo, pero manteniendo un id para separarlos según video. Se hizo el match entre descriptores y su palabra más cercana en el vocabulario, y se asignaron a un vector para cuantizar dichos votos. Este proceso se repitió para cada video, asegurándose de normalizar los histogramas según la cantidad total de descriptores presentes en dicho video, con el fin de que fueran comparables.

Posteriormente, se pasó a la etapa de evaluación de DURF, que entregó resultados muy bajos en comparación a los de Jiang [2012] para el mismo descriptor, por sí solo. Esto llevó a descubrir un error en el código para la persistencia del vocabulario, ya que se estaban almacenando, por cada video, únicamente los descriptores del último frame. Lo anterior se debió a que la matriz en vez de acumular el conjunto de DURF de cada frame, se sobrescribía en cada iteración. Por tanto, en vez de guardar los datos de aproximadamente 40 frames por video, sólo lo hacía con uno. Esto significa que el archivo que llegó a pesar 3 GB, realmente debía ser un orden de magnitud más grande.

Se hizo la corrección, y se comenzaron a calcular los valores nuevamente. Durante el transcurso del proceso se pudo estimar que el tamaño para la colección de entrenamiento completa sería mayor a 40 GB, por lo que se decidió hacer un cambio en la persistencia e interrumpir la ejecución. Se modificó el tipo de archivo sobre el que se escribirían los datos, pasando a ser un .txt, esto debido a que OpenCV requiere cargar en memoria el .yml para realizar sus operaciones. Si bien es cierto que 3 GB es bastante, el computador lo soportaba y se pudieron obtener resultados, sin embargo, un orden de magnitud mayor no sería posible, en el marco de la clase FileStorage. Además, se estimó que un .txt disminuiría el tamaño final.

Como método complementario para apoyar la reducción de tamaño del archivo (finalmente de 28 GB) y aumentar la eficacia del sistema, se permitió el uso de SURF en vez de DURF, ya que OpenCV presentaba un cálculo suficientemente rápido de SURF. La idea original detrás de DURF es disminuir tiempo de cómputo de descriptores y aproximar eficacia con muchos de ellos en ubicaciones fijas, mientras que SURF emplea una detección de puntos más complicada, pero entrega un número menor de vectores con mayor eficacia y representatividad de los datos. Además, se debe recordar que los autores del sistema tenían como objetivo un software capaz de clasificar en tiempo real videos subidos por usuarios, mientras que para efectos de esta memoria y sus objetivos, la demora en cómputo de descriptores es perfectamente aceptada. Cabe destacar que se agregó un archivo con metadatos, los que consideran para cada descriptor, a qué cuadrante pertenecen en la grilla de 2 x 2, y a qué sección pertenecen en una grilla de 3 x 1. De esta forma, al leer ambos archivos en paralelo, se pueden rescatar las ubicaciones espaciales, para facilitar el proceso de encontrar los subconjuntos que comparten una subzona.

Posteriormente, se alteró la interfaz para crear el vocabulario, de tal manera que pudiera

leer desde el archivo de texto. De esta forma, se creaba una matriz de OpenCV, y en cada fila se almacenaba un descriptor. Con este esquema, la ganancia en eficiencia no es muy grande, sin embargo, se adquiere transparencia en el proceso, se evita que el programa pueda suspenderse debido a la demanda de FileStorage, y permite, en caso de ser necesario, elegir algunas líneas del archivo, en vez de todas (subsamplear el conjunto). Se realizó una prueba, y se pudo apreciar que el computador era capaz de correr el proceso de carga de datos. Lamentablemente, al llegar a la etapa de clustering, con el objeto BOWKMeansTrainer, no fue posible continuar y se suspendió automáticamente la ejecución. Se concluyó que k-means en OpenCV, requiere matrices de mucho menor tamaño que la obtenida, y no puede seguir procesando intercambiando RAM por memoria externa. Además del problema anterior, se pudo notar que k-means no mostraba feedback en la consola durante sus iteraciones, por lo que un método con mayor visibilidad se hacía necesario para estimar tiempos de cómputo y decidir cambios de parámetros.

De esta manera se comenzó una nueva búsqueda, en este caso, por implementaciones de k-means que permitieran leer desde un archivo sus datos, o que no fuesen muy complicadas de alterar para este fin, en lo posible en un lenguaje sobre el que se tuviera experiencia, como C++, pero no se descartarían otras opciones al no cumplir con esta última condición. Es preciso recordar que, el objetivo detrás del uso del total de descriptores es para mantener la mayor cantidad posible de datos, y que k-means genere un vocabulario más representativo. Sin embargo, esto también depende de la cantidad de iteraciones, y es inherente a este problema un trade off entre tiempo de cálculo y eficacia. La posibilidad de reducir el tamaño de 30 GB (aproximadamente 50 millones de descriptores SURF) a un subconjunto, no se encontraba descartada, pero se deseaba darle prioridad a la colección completa de entrenamiento, antes de tomar la decisión de sub muestrear.

Se encontraron bastantes implementaciones, algunas provistas por software de minería de datos, como es el caso de R, y otras, en forma de programas en distintos lenguajes, de uso libre y publicados por científicos. Desafortunadamente, ninguna permitía leer desde un archivo las características que se deseaban clusterizar, y la única que sí cumplía con este requisito, era una implementación en PERL, que a pesar de parecer muy adecuada, tenía el inconveniente de manejar identificadores para los datos en formato de enteros. Es por esta razón que en la documentación se especificaba que el sistema no era capaz de indexar más de 2 GB, por tanto, este era el tamaño máximo de los archivos que podían ser procesados.

Aún manteniendo el objetivo de clusterizar toda la colección, se optó por implementar k-means en C++, con una interfaz para leer los archivos de texto. Se tuvo especial cuidado de que no se presentaran memory leaks, ya que el sistema, por sí solo, iba a requerir una interacción alta entre RAM y memoria externa, y un problema de este tipo significaría un riesgo de que la ejecución fuera suspendida, o el computador reiniciado. Cabe destacar que la versión de k-means que se implementó es sobre el algoritmo de Lloyd, esto debido a que en Jiang [2012] no se especifica alguno en particular usado por los autores, y resultaba ser el menos complejo desde el punto de vista algorítmico.

Se decidió que las estructuras se representarían mediante clases de la librería estándar de C++, como es el caso de vector y set. En particular, un descriptor SURF correspondería a un vector de float, y un conjunto de ellos, a un vector de objetos de este tipo. Para los clusters se utilizó set, de manera de aprovechar los métodos de comparación entre ellos, con respecto a la condición de término de k-means. Los elementos de cada set corresponden a los índices de los descriptores con en relación al objeto que los almacena, por tanto, son set de int. Además, existe un vector de clusters, con el tamaño 500 para el vocabulario.

Para realizar la lectura de los archivos se empleó la clase ifstream, y se leían en paralelo el archivo con metadatos y el de descriptores, de manera de filtrar, en caso de ser necesario, aquellos que pertenecían a una de las subzonas. Para los centroides se utilizó el mismo esquema de vector de vectores de float, y para seleccionar las semillas, fue necesario utilizar un generador de números aleatorios, para ello se recurrió a las funciones rand( ) y srand( ). Dichos números aleatorios permitieron elegir, y cargar en el objeto de centroides, 500 descriptores del conjunto completo. Se utilizó distancia Euclidiana para el ajuste de los centroides y los clusters con respecto a iteraciones sucesivas, y la condición de término correspondía a la no variación del conjunto completo de clusters, lo que se midió a través de la conjunción de comparaciones entre sets de una misma posición, en instantes sucesivos. Además, se añadió la posibilidad de elegir un número límite de iteraciones para dar término al algoritmo de k-means, en caso de no presentarse la convergencia mencionada previamente.

Una iteración del algoritmo se trataba de una búsqueda secuencial del centroide con menor distancia para cada descriptor, y posteriormente, la asignación del índice de dicho descriptor en el cluster correspondiente al centroide. A continuación, se analizaba el conjunto anterior de clusters, y se verificaba la posible convergencia, en caso de ocurrir ésta, se persistían los 500 centroides a un archivo que representaría el vocabulario para la subzona sobre la que se estaba trabajando. En caso de no terminar con el proceso, se limpiaría la estructura de clusters anteriores y centroides anteriores, y serían llenadas con los actuales, es decir, se recalcularían centroides, y se asignarían a la estructura. Para el cálculo de centroides se tomaban los índices del set correspondiente al cluster, y se buscaban en el objeto de descriptores, todos aquellos pertenecientes a dicho cluster. Con ellos, se obtenía el vector representante del centro de masa, y se trata del promedio de todos. Cabe destacar que, este vector generado, no necesariamente pertenece al conjunto, pues es un valor estadístico, y se puede considerar un descriptor artificial.

Una vez completa la implementación de k-means, se procedió a experimentar con ejemplos sencillos de clusterización, y para ello se crearon archivos de menor tamaño, con resultados conocidos, para corregir posibles bugs. Finalmente, se decidió pasar a la etapa de experimentación con la colección completa de descriptores de 28 GB. Desafortunadamente, el resultado no fue el esperado, ya que el sistema operativo eliminó automáticamente el proceso una vez alcanzados los 35 millones de descriptores cargados (de un total de 50). La causa del error se debió a límites de memoria, y a que Ubuntu cuenta con un módulo de protección para sus procesos, que en función del tiempo les asigna puntajes según la utilización de recursos, estos valores son comparados con un umbral, y al superarlo, se decide la detención de ellos.

Antes de continuar con la implementación, se decidió realizar un experimento que consistía en dejar de lado los descriptores de la distribución espacial de 1 x 1, ya que resultaban ser 50 millones, y testear el programa con una de las subzonas de la partición de 2 x 2, que se estimaba debían poseer la cuarta parte del total de descriptores, además de que se sabía que el proceso no sería interrumpido por el sistema operativo. Se dejó correr el programa por más de una hora, y no se recibió el feedback de cambio de iteración, por lo que se pensó que era demasiada la carga para el computador, y que a pesar de funcionar bien, el tiempo que requeriría realizar 10 iteraciones sería muy grande. Desafortunadamente, 10 iteraciones aún puede resultar poco, ya que se trata de 500 centroides, por lo que la convergencia posiblemente implicaría un número mayor de iteraciones. Es decir, el trade off entre mejorar la eficacia y aumentar la eficiencia era bastante alto. A partir de esto, se decidió buscar una manera de reducir el total de descriptores. Se consideró como alternativa realizar un muestreo aleatorio, y elegir una cantidad adecuada que sí pudiera clusterizarse en un rango de tiempo razonable. Sin embargo, se ideó una heurística que permitiera elegir valores que se creyó podían ser más representativos.

La primera etapa de la heurística consistió en un recorrido por la colección completa de descriptores, deteniéndose en cada video, y agrupando sus descriptores en el mismo tipo de estructura utilizada para k-means. Posteriormente, se clusterizaron los valores de SURF de cada video, y se generaron 200 centroides por video, los cuales se guardaron en un archivo de texto. Se escogió 200 debido a que una estimación por frame, al momento de haberlos calculado, entregó que cada uno aportaba en promedio esa cantidad, además de ser un número suficientemente menor como para obtener resultados en tiempos apropiados. En el caso de videos con menor número de descriptores, se almacenaron todos ellos y no se realizó clusterización.

El archivo generado resultó tener un tamaño aproximado de 500 MB, lo que cupo en memoria RAM, y facilitó en gran medida el trabajo. Una estimación revela que de 50 millones de descriptores, se pasó a contar con sólo 930 mil. A partir del archivo de 500 MB con descriptores artificiales, se realizó la etapa de clusterización normal, para elegir 500 centroides como vocabulario. Es decir, se obtuvieron centroides a partir de centroides, y no de descriptores reales. El proceso se repitió para las subzonas y se generaron 8 archivos de vocabulario. En este contexto, se modificó el programa que calculaba los histogramas, simplemente se leyó la colección completa de 28 GB y se realizó el conteo de votos para cada frame con respecto al vocabulario. Esto generó un histograma de 500 dimensiones por frame, lo que finalmente se promedió con el resto, entregando un histograma agregado para el video. Dichos histogramas agregados se almacenaron junto al id del video. Posteriormente, se concatenaron los histogramas de cada video por subzonas, presentes en los 8 archivos creados anteriormente. Esto significa, un histograma de 4000 dimensiones por cada video, que se escribieron a un archivo en el formato aceptado por los clasificadores, además de ser escalados con la herramienta de libsvm para mejorar los resultados de PmSVM.

Cabe destacar que el proceso de extracción de los 200 centroides por video tomó 12 horas para la distribución de 1 x 1, mientras que para las subzonas, se paralelizó y se consiguió calcularlas en 12 horas cada una, es decir, 36 horas fueron necesarias para la generación de

los archivos de centroides representativos. Además, en el caso de las subzonas, se permitieron más iteraciones de k-means al procesar los centroides por video, ya que se trataba de menos descriptores, y se aumentó a 300 la cantidad de centroides, a pesar de que muchos videos contenían menos que esa cantidad. El número de iteraciones era 5 en el caso de 1 x 1, mientras que en 2 x 2 y 3 x 1 se trataba de 7. Con estas dos medidas se logró que los archivos pesaran cerca de 700 MB, y que, idealmente permitieran mayor resolución y representatividad con respecto al total real de descriptores por subzona. En el caso de la clusterización para generar vocabularios, el tiempo fue de 24 horas, esto incluyendo paralelización de los procesos. Para los cálculos de histogramas del set de entrenamiento y de testing, en el caso de la partición de 1 x 1, fueron necesarias 12 horas, mientras que las divisiones espaciales de 2 x 2 y 3 x 3, requirieron en total 8 horas.

Para concluir, se pudo apreciar que la heurística funcionó mejor de lo esperado, y no fue necesario intentar con un muestreo aleatorio, ya que los resultados de PmSVM, en comparación a los de Jiang [2012], fueron considerados aceptables, y mucho mejores que los obtenidos con la implementación errónea mencionada en el comienzo de este apartado.

3.3.4. Módulo de clasificación SVM

Búsqueda de implementaciones

Se comenzó por buscar en la red una implementación de SVM que admitiera kernels introducidos por el usuario, o bien, que el kernel de tipo chi cuadrado ya estuviera incluido. Sobre este punto, cabe hacer una aclaración, en el marco teórico se define como kernel de chi cuadrado el que realmente se conoce como exponencial de chi cuadrado, esto es así ya que en el paper original de SUPER los autores lo manejan con dicho nombre. Sin embargo, existe una gran diferencia, ya que ambos son tipos de kernel distintos y existen implementaciones tanto de uno, como del otro, además el abuso de notación puede causar confusión.

Las implementaciones que se consideraron como alternativa para formar parte del módulo SVM fueron dos, una en Matlab [9], que cumplía con el soporte para el kernel exponencial, era multiclase y además admitía un esquema multimodal. La otra, consistía en un código en C++ llamado por los autores PmSVM y proveniente de Wu [2012], la cual no soportaba kernels no lineales, tales como el exponencial, pero admitía el kernel de chi cuadrado sin la potencia, y otro tipo de kernel llamado de intersección de histogramas (esto último será retomado posteriormente). Cabe destacar que hubo más implementaciones analizadas, pero de todas ellas, la de Matlab era la más completa, y en segundo lugar PmSVM, que además admitía gran posibilidad de personalización, ya que el código era mucho más sencillo que las combinaciones de archivos en C y .mex propios de Matlab. Entre las demás versiones de SVM encontradas en la red se encuentra libsvm, y sus variantes como svm-light y svm-multi,

[9] `http://www.maths.lth.se/matematiklth/personal/sminchis/code/libsvm-chi2.html`

todas ellas siendo más difíciles de extender y sin ciertos componentes clave para el sistema.

Debido a que la implementación en Matlab cumplía con todos los requisitos deseados, fue la escogida para integrar el módulo, y se tuvo en cuenta que la única dificultad técnica provenía de implementar la métrica de mAP, pues no sólo requería que el programa retornara las etiquetas de cada video de testing, si no que un ranking basado en los puntajes de SVM para cada video de consulta con respecto a cada clase. Esto implicaría un impacto en la funcionalidad de predicción, y manejo de la interfaz de código en C para métodos en Matlab, a través del API de MEX.

Para el manejo de diferentes características se cargaría cada una en un vector para las etiquetas, y una matriz con los valores de los descriptores o histogramas de ellos. Posteriormente, cada fila de cada matriz se concatena con la fila correspondiente de la siguiente, en el caso de los vectores de etiquetas basta con uno solo. De esta manera se cumple con la interfaz de la implementación que entrega un único modelo SVM a partir de un solo vector de etiquetas, una sola matriz de instancias y un arreglo de opciones, entre las cuales se debe especificar la cantidad de dimensiones que ocupa cada característica para poder distinguirlas internamente, y los pesos asociados en la fusión tardía de puntajes, así como un valor para la constante gamma que aparece en la fórmula del kernel exponencial de chi cuadrado. En este caso, todos los pesos corresponden a 1.0 para obtener el efecto late average fusion, y gamma se utilizó como 1.0 ya que no se especifica en Jiang [2012] un valor, no obstante se probó con modificaciones, aunque no se apreciaron mayores diferencias. El modelo retornado equivale a múltiples modelos para cada característica, ya que internamente se maneja de esta manera, y no significa una fusión temprana de datos. Antes de realizar testing con el sistema completo, es decir, con los 7 tipos de descriptores, se llevó a cabo el entrenamiento considerando SIFT, y GIST, cada uno por separado dejando de lado la componente multimodal, y enfocándose solamente en las accuracies obtenidas, pues antes de implementar mAP era necesario corroborar un buen funcionamiento del módulo SVM.

Los resultados indicaron que, a pesar de variar gamma, las etiquetas predichas para todos los histogramas SIFT de testing fueron de la clase 0, que corresponde a los ejemplos negativos de la colección, es decir, aquellos que no poseen eventos presentes y que cumplen la función de distractor para aumentar la capacidad de generalización del modelo entrenado. Debido a que dicha clase se encuentra sobre muestreada, con cerca de 1000 entradas, el valor de accuracy fue aproximadamente de un 22 %, pues son 4658 videos de testing. A continuación, se repitió el proceso con GIST y se obtuvo exactamente el mismo resultado, todas las etiquetas iguales a 0 y accuracy de 22 %. Se concluyó que la implementación de Matlab no estaba completa, y alguno de sus componentes inducía a errores. Se descartó que fuera producto del tamaño de los descriptores, ya que, a pesar de que los histogramas de SIFT cuentan con 5000 dimensiones, GIST posee únicamente 512. Finalmente, se intentó con MFCC y STIP, y se llegó a los mismos problemas.

Analizando con ejemplos de clasificación presentes en la red, e incluso con los mismos que provee la descarga del paquete en Matlab, se obtuvieron resultados consistentes, y no se

apreció el mismo inconveniente que con los datos de CCV. Por esta razón, se pensó que las dos variables que podían estar influyendo eran la cantidad de entradas de training y testing, ya que datasets mucho más pequeños funcionaban correctamente, o bien, el problema de que los descriptores empleados en CCV son histogramas con una gran cantidad de ceros. Por otra parte, el tiempo que tardó Matlab en completar la clasificación de SIFT llevó a concluir que, en caso de hacer testing fusionando características, el entrenamiento tardaría gran cantidad de tiempo, ya que las 5000 dimensiones de SIFT se deberían sumar a las de los demás descriptores. Esto además incurre en el riesgo de que Matlab no posea suficiente memoria como para realizar el proceso.

Frente a los inconvenientes mencionados previamente, se tomó la decisión de evaluar la implementación de PmSVM, a pesar de que implicaba un costo de personalización y extensión del software. Además, había que considerar que PmSVM sólo soporta kernels lineales, por lo que, en principio, se requiere una modificación de alto impacto para poder agregar la componente exponencial faltante. Debido a lo anterior, se agregó como restricción la utilización de kernels lineales para testear PmSVM, y en caso de obtener resultados cercanos a los de Jiang [2012], conservar dicha implementación.

Implementación definitiva

La primera etapa de la evaluación consistió en ejecutar el comando de compilación entregado en la documentación [10] para generar un único ejecutable, con opciones de realizar entrenamiento directamente o para emplear crossvalidation y obtener ciertas métricas respecto a eso. Esto último no es necesario ya que CCV posee set de training y testing bien definidos, y los autores no declararon haber potenciado sus modelos con métodos como el anterior. Por su parte, la opción para entrenar un modelo SVM es sencilla, ya que recibe dos archivos de texto, uno con los datos de entrenamiento y otro con los datos de testing. Con el primero genera el modelo, y una vez terminado este proceso, carga el segundo y realiza las consultas al modelo. Finalmente, entrega dos valores agregados, que tienen relación con la matriz de confusión, uno de ellos es overall accuracy y el otro es average accuracy, los que permiten identificar qué cantidad de aciertos tuvo el clasificador sobre el total de consultas.

Cabe destacar que el program descrito no está pensado para un dataset tan grande como CCV, ya que, cada vez que se desea entrenar un modelo, se comienza de nuevo y el que existía sólo se mantuvo en memoria hasta el final del proceso. Además, sólo está permitido el uso de un tipo de característica, y para un esquema multimodal sería necesario crear archivos de texto con todos los valores de los descriptores concatenados, y como PmSVM no recibe los parámetros de dimensiones de cada uno como para poder separarlos, se estaría realizando fusión temprana. De esto se desprende que, para ajustar PmSVM a las necesidades del problema se necesitaba añadirle funcionalidades, específicamente, métodos para persistir y cargar modelos, además de intervenir el código para realizar evaluación de los diversos

[10] `https://sites.google.com/site/wujx2001/home/power-mean-svm`

modelos, y en vez de retornar una etiqueta para cada uno y luego escoger la que posea más votos, se deben ponderar los puntajes de SVM y a partir de ellos retornar una sola etiqueta.

Debido a que se estaba considerando la posibilidad de utilizar PmSVM, se empleó el ejecutable compilado, y se visualizaron los valores de accuracy obtenidos para algunos descriptores. Como se mencionó previamente, el programa no estaba pensado para un dataset tan grande, por tanto el tiempo de entrenamiento se extendía por media hora para los descriptores con más dimensiones (sobre 5000). Finalmente, se pudo apreciar, que había un error conceptual al comparar accuracies con precisiones, pues son métricas diferentes, sin embargo, una leve intrusión en el código permitió acceder a la matriz de confusión y calcular precisión mediante las filas y columnas de ella. Con estos nuevos valores se recompiló y se obtuvieron números cercanos para SIFT, MFCC y STIP, con respecto a la contraparte original. Es por esto que se decidió proceder con la intervención y añadir mayores funcionalidades hasta alcanzar la suficiente como para integrar PmSVM al móduo SVM del sistema.

El primer paso fue crear lectura y escritura para los modelos SVM, para ello se examinó el header de PmSVM y se encontraron dos estructuras importantes, una es problem y la otra model. La primera, se encarga de guardar las variables que definen al problema de clasificación, es decir, número de clases, dimensión de los vectores, cantidad de instancias, y las etiquetas usadas para identificar cada descriptor. Esto último, se debe a que el programa utiliza enteros comenzando desde cero, y existe la posibilidad de que un usuario especifique etiquetas con valores negativos, o no correlativos, por ello se renombran y se mantiene un registro interno para convertir al dominio del usuario en caso de ser necesario. Es tarea de **problem** generar un modelo, ya que posee métodos para el entrenamiento. La segunda estructura almacena las variables que representan al modelo SVM, éste se caracteriza por permitir más de dos clases, es decir, internamente se compone de modelos binarios de uno contra el resto. Además, admite diferentes kernels lineales, los que se especifican con un parámetro **p**, que en caso de ser -1 se obtiene chi cuadrado, y con -8, intersección de histogramas. También se guardan otras variables como el total de clases, y parámetros que ayudan a determinar las características del espacio generado por los vectores de soporte.

Una vez hecho este análisis, se creó una función para escribir los atributos de un objeto model a un archivo de texto mediante un formato sencillo, donde en cada línea se guardaba uno distinto. En el caso de un vector, se utilizaba una sola línea y cada coordenada se almacenaba con separación de un espacio. Cabe destacar que PmSVM, en su código por defecto, se encarga de dividir el comportamiento en los objetos mencionados, y la experimentación se lleva a cabo únicamente en model. Por tanto, con un archivo de texto es suficiente para el nuevo ejecutable, que no requiere la utilización de **problem**, y comienza desde el punto en que el clasificador ha sido entrenado. Esta característica es importante para garantizar un sistema multimodal.

Para verificar la correctitud de la nueva funcionalidad, se agregaron los métodos al archivo .cpp y sus firmas al header, con lo que se recompiló el ejecutable básico de PmSVM. Dicho ejecutable fue modificado, y se interrumpió la etapa en la que trabaja con el objeto model,

que **problem** generó, para persistirlo a un archivo de texto, y luego, cargarlo en un nuevo model para continuar con la ejecución. Los resultados fueron favorables, y quedó en evidencia que no se necesitaban variables adicionales, que hubiesen estado explícitas en el header. Con esto se dio por terminada la persistencia de modelos SVM, considerando que en el sistema final sería un archivo para cada descriptor y debían ser cargados simultáneamente.

En la siguiente etapa, se deseaba implementar la capacidad para cargar más de un modelo SVM, y fusionar, de manera tardía, sus resultados de clasificación, para así obtener un esquema multimodal. Para ello se dividió PmSVM en dos ejecutables, el primero, una versión modificada del original, que escribía en un archivo el modelo entrenado, y el segundo, que creaba por cada archivo de texto ingresado, un objeto model y realizaba la fusión. Esto implica que el módulo de clasificación, propiamente tal, se encontraba cubierto, pues la arquitectura del sistema indica que las características se entrenan por separado y se generan clasificadores SVM. Esto significa que todo el trabajo realizado sobre el segundo ejecutable, formó parte de la etapa de fusión de datos.

Dado lo anterior, comenzó el proceso que abarcó más tiempo de implementación en esta memoria, y que consiste en calibrar el sistema para emular los resultados de Jiang [2012]. La extensión de esta instancia se debe a que, a pesar de conocerse los valores de mAP para cada descriptor por sí solo, y para diversas combinaciones de ellos, no se tenía claridad sobre la manera de calcular esta métrica, y hubo diversos grados de libertad que generaban ruido. A continuación, se mencionarán los problemas y sus soluciones conforme se fueron encontrando.

La primera versión del ejecutable fue probada con los descriptores provistos en la descarga de CCV: SIFT, STIP y MFCC. Esto se debe a que sus histogramas fueron calculados por los mismos autores y publicados de manera de transparentar su esquema Baseline de reconocimiento de videos. Además, los valores de mAP de cada uno, y de su conjunto se encuentran tabulados en el paper, y por tanto, se decidió que eran la mejor fuente de información para calibrar el sistema. Esta primera versión consideraba una sola etiqueta de clase para cada video, esto se debe a que en el archivo que indica a qué clase pertenece cada video, existen videos con múltiples clases. Esta característica se suele abordar como un problema donde el ideal del sistema es que sea capaz de entrenarse conociendo estas instancias, y adquiriendo la capacidad para diferenciarlas, de tal forma que en un objeto de testing que posea exactamente las mismas clases, el sistema las reconozca todas. Esto se denomina multi-label. Sin embargo, en Jiang [2012] no se declara esta modalidad de funcionamiento, y se muestran resultados agrupados por clase, entendiendo como tales, las 20 mencionadas previamente. Por tanto, se decidió elegir una sola etiqueta por instancia para realizar las primeras pruebas. Los resultados se midieron con overall accuracy y sin combinar características, ya que aun no se implementaba esta última funcionalidad. Se pudo apreciar que el sistema tenía un valor de eficacia muy bajo para los descriptores individuales, y se pensó que era necesario completar la parte restante para tener una idea del conjunto de vectores.

La siguiente versión, se obtuvo cargando los modelos de los tres tipos de descriptores, y para cada instancia de testing, se retornaba una etiqueta por cada modelo. Posteriormente se

consideraba cada una de las tres como un voto para la clase respectiva, y en caso de que una de ellas obtuviera mayoría, se asignaba esa etiqueta final, en caso contrario, se retornaba, por defecto, la decisión del modelo de SIFT, que según los datos de Jiang [2012] era el más eficaz. Nuevamente, se obtuvo un valor muy bajo de overall accuracy, por lo que se decidió hacer cálculos de precisión, ya que era posible que accuracy estuviera correcto y que aun así mAP entregara valores altos, como en el paper.

Para la siguiente iteración se utilizó la definición de precisión con respecto a la matriz de confusión, diferente a la de information retrieval basada en rankings. Con esta métrica se calculó un promedio y se obtuvo un valor de AP. Analizando el problema se descubrió que había un error conceptual, ya que, dado el escenario descrito, era imposible llegar a un valor de mAP, pues sólo había uno para AP. De esta forma, se concluyó que la manera de medir el sistema en Jiang [2012] era diferente, y debía estar basada en alguna clase de ranking. Se inspeccionó con mayor cuidado el paper, y se descubrió que la manera empleada por los autores para evaluar el reconocedor de eventos en video era muy similar a la de las competencias de information retrieval. Para ello, se calculaban para cada video de consulta, los puntajes SVM de cada clase, es decir, se obtienen valores análogos a la probabilidad de que dicho objeto pertenezca a una de ellas. Posteriormente, el puntaje del video de testing se ingresa en un ranking de la clase correspondiente, y se repite para todos los demás puntajes. El proceso se vuelve a realizar con el resto de los videos.

De lo anterior se obtiene un ranking con el orden de cada video de testing para una clase en particular, y es posible calcular AP para ella. Finalmente, se obtiene mAP como promedio de los valores de AP por clase. Se pudo notar que el enfoque de calcular una etiqueta basada en sus votos estaba incorrecto, ya que el sistema debería sumar los puntajes de SVM de cada característica y retornar aquella con más probabilidad. Dado el contexto anterior, se estudió el código, y se descubrió que en el funcionamiento del ejecutable básico de PmSVM, se requiere un arreglo con los puntajes para cada video, el que puede ser llenado mediante la función de predict, y se le entrega como parámetro. Esto simplifica la tarea, ya que se realiza para todos los descriptores y se pueden sumar los puntajes y añadir el video a los rankings de la clase correspondiente. Siguiendo esta línea se programó la lógica necesaria y se obtuvo una estructura con la información para realizar los cálculos de mAP.

En esta versión del ejecutable se obtuvieron resultados de SIFT, STIP y MFCC bajos, y su conjunto seguía siendo mucho menor al valor del paper. Por lo que fue necesario estudiar otros focos de error, como es el caso del problema multilabel. Se volvió a entrenar el sistema, con la diferencia que ahora se agregaron etiquetas para cada combinación de clases, por ejemplo, si un video pertenece a **Dog** y a **Basketball** se le asigna un id, de manera que los clasificadores aprendan qué características tiene un video que involucra a ambas, y además, posea un identificador como etiqueta que permita reconocerla en un video de testing. Este proceso es el que se emplea para convertir un clasificador como PmSVM en multilabel.

Los resultados fueron mucho más bajos que los de las versiones anteriores. La cantidad de clases nuevas superaba de gran manera las 20, y algunas de ellas no tenían ejemplos de testing

asociados. Además, habían videos en la colección de testing que pertenecían a combinaciones de nuevas de clases, y por tanto, el clasificador no estaba preparado para ellos. Esto llevó a concluir que el problema no había que considerarlo como multi-label, y que el dataset no estaba diseñado para ello.

Posteriormente, se recordó que en la documentación de PmSVM se menciona que dicha herramienta funciona de mejor manera al recibir los datos de entrenamiento y testing escalados a valores entre 0 y 1. En casos de clasificación con un único tipo de descriptor o histograma, resulta menos importante esta sugerencia. Sin embargo, al fusionar los puntajes de predicción de clasificadores SVM para diferentes dominios de vectores característicos, uno de ellos puede predominar por sobre el resto, debido a los rangos entre los que se mueven sus puntos. Es por esto que, al añadir multimodalidad al ejecutable, el escalamiento es necesario, y permite realizar una categorización equitativa, otorgándole a cada puntaje de SVM un mismo ponderador. En particular, STIP fluctuaba entre 0 y 250, mientras que MFCC y SIFT lo hacían entre 0 y 1.5, con los que STIP dominaba las predicciones, anulando el efecto de los otros clasificadores.

Para solucionar el problema mencionado previamente, se empleó el ejecutable scale de la librería libsvm, ya que se sugería esto mismo en la documentación de PmSVM. Y se modificó el código de scale, para que escribiera todos los valores de los descriptores al nuevo archivo y dejara de omitir los ceros. Esta propiedad de sparse, puede ser decodificada por libsvm, sin embargo PmSVM no admite esta modalidad. Con esta modificación los resultados se acercaron bastante a los del paper, pero no llegaron a ser los mismos. Es por esto que se retomó la idea de las múltiples etiquetas y se pensó que los autores las usaban como alternativas para el sistema, es decir, en caso de que un video posea más de una, si el reconocedor de eventos elige cualquiera de sus etiquetas, entonces se considera exitoso, y por tanto aumenta los valores de eficacia. Se programó esta última parte y se llegaron a resultados suficientemente cercanos a los originales, por lo que se pensó que las diferencias existentes se debían a la naturaleza de los kernels empleados, más que a un problema de implementación o a un error conceptual.

Finalmente, se decidió conservar esta implementación por sobre las demás que fueron analizadas, dado que su eficacia se aproxima bastante al experimento de Jiang [2012], y se aprecian las relaciones de orden en el desempeño de diferentes descriptores. Además, se refactorizó el código para admitir una cantidad variable de modelos SVM, ya que en las versiones de prueba se mantenía una cantidad fija de tres. También se pudo apreciar que la métrica de mAP de los autores está correlacionada con AP obtenido mediante matriz de confusión y con overall accuracy, pues un aumento en alguno de ellos indica que los demás también serán mayores.

3.3.5. Módulo de clasificación K-NN

Para la implementación de este clasificador se utilizó gran parte del código proveniente del sitio: [11]. Esta rutina se empleó como template, ya que hubo que hacer bastantes modificaciones para cumplir con los requerimientos del sistema. Inicialmente, existía soporte para realizar la clasificación entregando un valor para k, y se manejaban etiquetas y vectores de características con punteros a double. Además, se requería código del mismo autor, específicamente, una función llamada squared distance, que se usaba para las comparaciones. Buscando en la red, se encontró la implementación y se añadió directamente la función al .cpp del clasificador. Cabe destacar que, que dicho método no era complicado de programar, sin embargo, resultó mucho más conveniente haberlo encontrado, pues respetaba directamente las interfaces propuestas por el autor.

El primer paso fue modificar las llamadas a punteros, por vectores de la librería estándar de C++. Se probó el módulo con dichos cambios y se consideró que funcionaba correctamente. A continuación, se agregó un método para leer el mismo formato de archivos de PmSVM y asignar sus características a vectores de double. Cabe destacar que el entrenamiento de un clasificador K-NN consiste en cargar en memoria los histogramas o descriptores de la colección de training, tarea que cumple dicho método. Posteriormente, se añadió un ciclo que recorriera los histogramas de la colección de testing, y se aprovechó la función de lectura, para cargarlos en vectores. Por tanto, la iteración se realizaba con respecto a estructuras en memoria, y se contaba con todas las características del set de testing de inmediato.

Se llamó a la función predict asociada al clasificador con cada vector de consulta, y se probó con diferentes valores de K. Los resultados indicaron que este módulo funciona más lento que SVM, y esto se debe a que K-NN es lazy en su generación de modelo, pues en realidad, su modelo son los datos de entrenamiento cargados en memoria, y la distancia Euclidiana al cuadrado (se evita calcular la raiz cuadrada), la que se encarga de elegir las etiquetas. Por otra parte, SVM posee un modelo liviano que puede almacenarse una vez completada la etapa de training, y con menores cálculos es capaz de estimar una clase para un objeto de consulta.

A continuación, se verificó la correctitud del programa, y se añadió soporte para multimodalidad, ya que, hasta este punto el clasificador permitía un único tipo de descriptor. Para esto se agregaron estructuras compuestas de C++, como por ejemplo, vectores de vectores de double. Se modificó el método predict, para iterar por sobre todas las características ingresadas, y por sobre los k vecinos dentro de cada una, para poder estimar la etiqueta adecuada. Inmediatamente la complejidad del programa aumentó, y se volvió aún más lento el clasificador K-NN, sin embargo, su eficacia mejoró.

Posteriormente, se comenzaron a añadir las métricas, pues se requería un valor de mAP para comparar con SVM. En esta etapa se descubrió un problema que no había sido previsto,

[11] `http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=9663&lngWId=3`

el cual consiste en la falta de un método estándar para calcular mAP con respecto a un clasificador K-NN. En el caso de SVM, se utilizaban los puntajes para construir un ranking por clase, por tanto, se pensó que debía utilizarse una metodología similar para K-NN, sin embargo, no se encontró apoyo científico que describiera un procedimiento aceptado por la comunidad, por tanto, cualquier manera para calcular mAP que se pudiera emplear, sería propia, no confiable, e incluso no comparable a SVM.

Algunas alternativas propuestas guardaban relación con las distancias cuadradas que utiliza K-NN, y visualizarlas como puntajes para formar un ranking. Sin embargo, esta idea no incluye el valor de k, por lo que para todo experimento que se realizara, el resultado sería el mismo. Es decir, no entregaba un valor propio del sistema. Otra versión tenía relación con el tamaño de k, y los k vecinos más cercanos para una consulta, tomando la cantidad de votos, para las clases que aparecieran entre ellos, como puntaje para formar el ranking. Mientras que las clases que no se manifestaran entre los k vecinos, se asignaría un puntaje cero para ese video consulta en el ranking de dicha clase. Esta alternativa parecía mucho más confiable, sin embargo, se desestimó su uso, puesto que aparecerían muchas instancias con valor cero en los rankings, y el orden para agruparlas afectaría el rendimiento.

A manera de aclarar estas decisiones, es importante notar que para comparar los sistemas se requiere que las métricas se encuentren en términos comunes y sean estándar. Emplear algunas de las alternativas mencionadas previamente, implicaría resultados sin un soporte científico, y las comparaciones no serían válidas. Pues se puede argumentar que la métrica empleada para K-NN beneficia el sistema, con el afán de parecer superior a SVM, o vice versa. Finalmente, se decidió que lo más correcto era comparar los sistemas en términos de sus matrices de confusión, y las métricas como accuracy que se pueden obtener de ellas. En el caso de precisiones, es posible utilizar la versión calculada a partir de dichas matrices. Por tanto, se implementó un método que expusiera la matriz de confusión del módulo K-NN, y que entregara las métricas apropiadas.

3.3.6. Tabla de parámetros

Tabla 3.2: Tabla de parámetros.

Componente	Parámetros	Implementación base
SIFT, STIP, MFCC	No requieren parámetros.	Archivos en la descarga de CCV. `http://www.ee.columbia.edu/ln/dvmm/CCV/`

Componente	Parámetros	Implementación base
CM	Imagen dividida en 5 x 5 (25 subzonas). Espacio de color L*a*b (3 canales). Por canal y subzona se calculan los 3 primeros momentos de las intensidades de gris. Se agrupan los datos en un vector de 225 dimensiones (25 x 3 x 3).	Implementación propia.
LBP	Ventana deslizante de 3 x 3 (8 vecinos por cada pixel). Códigos binarios de largo 8, un bin de histograma por cada código. Histograma cuantiza la ocurrencia de los códigos alrededor de todos los píxeles (256 dimensiones).	`https://github.com/nourani/LBP`.
GIST	Imagen en escala de grises. Se usan 8 orientaciones, una grilla de 4 x 4 y 4 posibles escalas. Vector resultante de 512 dimensiones (8 x 4 x 4 x 4).	`http://lear.inrialpes.fr/software`).
SURF	Se requiere un detector de puntos de interés que funcione con el algoritmo del hessiano. Se debe fijar un umbral, de tal forma que sólo los puntos cuyo hessiano supera dicho umbral son seleccionados (se utilizó 500 en OpenCV). Se requiere un número de octavas de pirámide Gaussiana (se usó 4), y una cantidad de imágenes entre dichas octavas (se utilizó 2). Finalmente, se requiere un extractor de SURF.	Se implementó en OpenCV, por lo que los parámetros mencionados, a excepción del umbral, vienen por defecto con la funcionalidad asociada a SurfFeatureDetector y SurfDescriptorExtractor (clases empleadas).
SSD	Se requiere un número impar como tamaño de los parches que se usarán en el cálculo de SSD (se utilizó 5). Además, se necesitan el radio del descriptor (se empleó 40), el número de bins del radio (se usó 3) y el número de bins para el ángulo (se utilió 12). Los valores mencionados vienen por defecto, y corresponden al artículo Chatfield et al. [2009].	A pesar de que se eliminó del sistema, su implementación fue extraída de `http://www.robots.ox.ac.uk/~vgg/software/SelfSimilarity/`.
Frames analizados por video	Se consideró un frame cada dos segundos.	Implementación propia en OpenCV.

Componente	Parámetros	Implementación base
Módulo SVM	Se utilizó un kernel de chi cuadrado (en vez de exponencial de chi cuadrado), se requirió escalar los histogramas mediante la herramienta **svm scale** de libsvm.	Implementación tomada de `https://sites.google.com/site/wujx2001/home/power-mean-svm`.
Módulo K-NN	Se probó con K entre 1 y 15, considerando únicamente números impares.	Implementación basada en `http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=9663&lngWId=3`.
Generación de Vocabularios	Se utilizó un valor de 500 para k-means. Cada imagen se consideró 8 veces, esto debido a las tres divisiones espaciales empleadas, la primera correspondiente a la imagen original, la segunda, que genera cuatro subzonas (grilla de 2 x 2) y la tercera de tres (grilla de 3 x 1). Se obtuvieron vocabularios de tamaño 500 por cada subzona, sobre toda la colección.	Implementación propia.
Generación de Histogramas	Para cada subzona de las divisiones espaciales se generó un archivo de histogramas por video, para ello se recibió su vocabulario asociado. Posteriormente todos los histogramas fueron concatenados.	Implementación propia.
Métricas	Reciben una matriz de confusión (generada al hacer las consultas), y en el caso particular de mAP, se recibe una estructura que asocia para cada clase un ranking con todos los videos ordenados según su puntaje de clasificación.	Implementación propia.

Capítulo 4

Discusiones y experimentación

En este capítulo se discuten los problemas de mayor complejidad que surgieron durante el desarrollo del reconocedor de eventos en video. El objetivo es señalar puntos críticos al momento de implementar un sistema similar. Posteriormente, se presentan los resultados de la evaluación de eficacia, y se comparan los sistemas Baseline y Modificado.

4.1. Dificultades en la implementación del sistema

4.1.1. Creación de vocabularios

Para la implementación del sistema reconocedor de eventos en video fue necesaria la condensación de descriptores en conjuntos de menor tamaño denominados vocabularios, que a su vez permitieron disminuir el volúmen de información a un único histograma por cada video. Esto en el contexto de los descriptores locales, ya que para efectos de descriptores globales bastó con promediar los resultados de los frames de cada video. Para realizar la extracción de vocabularios se optó por utilizar el algoritmo de k-means, que pertenece al conjunto de herramientas de clusterización, es decir, agrupar los datos en conjuntos cercanos entre sí. Esta cercanía se mide en base a distancias entre vectores, siendo escogida la distancia cuadrada.

Durante el desarrollo de esta componente, se pudo apreciar una gran dificultad desde el punto de vista de la cantidad de datos con la que se estaba trabajando. Específicamente se trataba de los valores de SURF, ya que los demás histogramas de descriptores locales habían sido obtenidos al descargar CCV, o se descartaron del análisis (SIFT, MFCC, STIP,

SSD). El volumen era de 28 GB, que correspondían a 50 millones de instancias de SURF (64 dimensiones), generadas a partir de los 4659 videos de entrenamiento. La problemática surgió debido a que los autores del sistema original habían realizado la clusterización considerando todas esas instancias, y en base a ellas construyeron los vocabularios (para otras orientaciones espaciales eran menos de 50 millones pero se mantenían en el orden de 10 millones). Por tanto, para mantener la implementación fiel a su contraparte original, era necesario idear una manera para procesar ese volumen de datos sin excluir instancias de vectores.

Desafortunadamente, de las diversas maneras de abordar este problema que se analizaron, todas incluían un alto intercambio entre RAM y memoria externa, ya que 28 GB es mayor a los 4 GB de RAM del computador en el que se realizó el trabajo. Es preciso recordar que cada iteración de k-means implica recorrer los millones de descriptores, y calcular distancias entre ellos y los 500 centroides del vocabulario preliminar. Por tanto, para realizar el proceso de la misma manera que el original, era necesario buscar una manera de sortear este obstáculo.

Una opción hubiera sido una constante lectura y escritura a un archivo para intercambiar páginas de datos que cupieran en RAM. Esto significa elaborar un programa que contempla el problema y explícitamente realiza las transiciones de memoria interna y externa. Técnicamente es más costoso de implementar que la opción de dejar que el sistema operativo lo haga, es decir, forzar la utilización total de RAM, manteniendo una estructura con todos los datos, y que internamente se manejen las transiciones. Esta última idea, por ser menos costosa se prefirió. Sin embargo, se encontró el inconveniente de que el sistema operativo rechazaba la ejecución completa de la carga de descriptores en memoria, ya que a pesar de escribir los valores en la estructura y realizar los intercambios de externa e interna, el proceso era interrumpido cerca de los 35 millones de instancias, porque un algoritmo propio de Ubuntu lo consideraba potencialmente peligroso, basado en un esquema de puntuación acorde al uso de los recursos.

Lo anterior llevó a plantear la posibilidad de paralelizar k-means, sin embargo, no se contaba con más equipos, y se descartó emplear los de la universidad, pues se necesitaba cargar los 28 GB y gestionar la comunicación entre todos. Por otra parte, mantener una sola máquina implicaría compartir RAM en threads, y la problemática afectaba mayormente al uso de memoria, por sobre el tiempo de cómputo de las distancias entre vectores, es decir, la necesidad era una mayor capacidad de almacenamiento, por sobre la eficiencia de cálculo. Por tanto, se desestimó la opción del paralelismo, y se volvió a considerar la escritura y lectura asistidas, para gestionar el uso de memoria.

Una nueva problemática surgió en relación al tiempo que demoraba una iteración del algoritmo de clustering. Experimentos con 10 millones de datos indicaron que una de ellas tardaría por lo menos 5 horas, y esto no asegura convergencia de k-means, pues se trata de millones de datos que deben reducirse a 500, con lo que es de esperar una mayor cantidad de iteraciones necesarias.

Todo lo expuesto con anterioridad en este apartado, llevó a descubrir la necesidad de

un sistema capaz de manejar un volumen de datos de 28 GB, que a la vez pudiera hacerlo eficientemente. Se piensa que una alternativa que cumple con estas características es un cluster de computadores que paralelicen cálculos de distancias, y que a su vez compartan memoria externa. Sin embargo, en el caso de un sólo computador, fue necesario implementar una heurística propia que redujera los datos a 3 GB, mediante dos etapas de clustering, pero no existe certeza de la correctitud de dicha estrategia, pues introdujo datos artificiales. Se cree que este esquema podría producir sesgo en los modelos generados por los clasificadores, y producir un sistema sobre ajustado al dataset de CCV, que no necesariamente resulte generalizable a otros videos de consumidor.

La heurística empleada no se basa en literatura relacionada a machine learning, y fue implementada desconociendo sus consecuencias en el sistema. Incluso, a pesar de que los resultados indicaron que los histogramas de SURF obtenían métricas mejores que, por ejemplo, MFCC, no está claro que sea una estrategia recomendable, ya que desde un punto de vista teórico, un enfoque libre de la problemática de corromper los datos, es la elección aleatoria de una muestra de menor tamaño de los 28 GB, como por ejemplo, 3 GB escogidos al azar. Cabe destacar que esta alternativa tampoco asegura un valor alto de eficacia.

Finalmente, además de un enfoque en paralelo con múltiples equipos, se hace deseable una solución algorítmica que resuelva este problema, ya sea reduciendo los datos sin la introducción de sesgos, procesándolos de manera más eficiente, o permitiendo la utilización total de ellos en un sólo computador.

4.1.2. Costo de extracción de descriptores (problemas con SSD)

Para calcular los descriptores de Self Similarities, se utilizó una implementación creada por los autores del artículo en el que se propone dicho tipo de descriptor. Por tanto, se consideró correcta, y no se indagó más sobre posibles alternativas. Una vez fija la implementación, además de los parámetros adecuados según Jiang [2012], fue posible realizar un análisis del tiempo de cálculo de SSD para una imagen de ejemplo. Estos experimentos llevaron rápidamente a la conclusión de que era necesaria buscar una manera de hacer la extracción de SSD más eficiente.

En primer lugar, los autores recomendaban un paso de normalización que se incluía como una función en el código. Esto para filtrar descriptores que no cumplieran criterios que ellos habían determinado en su investigación, además de permitir la comparación entre imágenes de diferentes tamaños. Sin embargo, esta función tardaba más de media hora para imágenes de 320 x 240, lo que la hacía muy costosa si se deseaba realizar para los frames escogidos por video, razón por la cual se descartó del análisis. En segundo lugar, la extracción de descriptores era cercana a un minuto por imagen de 320 x 240, con lo que para un video (en promedio 40 frames), pasaba a ser un proceso de 20 minutos. Realizando estimaciones se concluyó que tomaría meses procesar la colección completa, incluso paralelizando la ejecución.

Esto llevó a eliminar por completo los descriptores SSD del sistema reconocedor de eventos en video.

A partir de las problemáticas anteriores, se descubrió la existencia de algoritmos de extracción de características costosos de calcular, que significan grandes dificultades técnicas para ser incluídos en sistemas con un volumen de datos como el de CCV. Además, se concluyó que la solución a esto implica mayor poder de procesamiento, como por ejemplo, un conjunto de máquinas trabajando simultáneamente. Se cree que para un tamaño menor de datos, SSD es perfectamente utilizable, sin embargo, es preciso recordar que en el contexto de análisis de contenido en videos, el número de imágenes es inherentemente grande, pues cada uno está formado por una secuencia de ellas.

Por otra parte, se plantearon alternativas, como la reducción del tamaño de los frames, o sencillamente, considerar menos de ellos por video. No obstante, los objetivos del trabajo de título se centraban en la comparación de módulos de clasificación, por sobre la implementación y extracción de todos los descriptores utilizados en el sistema original. No se conoce el impacto real de la exclusión de SSD del análisis, ya que son múltiples las variables que influyen en el problema de clasificación, por ejemplo, es posible que un cierto tipo de descriptor funcione mejor con el módulo K-NN, y de esta forma entregue una ventaja con respecto a SVM, o vice versa.

Finalmente, cabe destacar que la gestión necesaria para el cálculo de SSD en la colección completa, hubiera implicado un trabajo mucho mayor, que escapaba a los límites de esta memoria, ya sea resolviendo el problema mediante hardware, como ideando estrategias para abordarlo.

4.1.3. Interpretación del problema de clasificación

Una temática que no fue abordada con mucho detalle en este documento, es sobre el problema de etiquetado de los videos de CCV. Entre los archivos provistos al descargar el dataset, se encuentran dos de ellos que indican las clases a las que pertenece cada video (un archivo para training y otro para testing). Inspeccionando su contenido se puede apreciar que existen dos casos de interés. El primero, se trata de videos que no presentan una de las posibles 20 clases, y reciben la etiqueta 0, mientras que el segundo posee más de una etiqueta.

Debido a la situación anterior, fue necesario investigar la forma correcta de interpretar la información. Para ello, se le preguntó a los autores del sistema original, y se descubrió que CCV contemplaba una clase extra para representar ejemplos que no pertenecen a ninguna de las otras 20, lo que corresponde a la clase 0 que se mencionó. Por otra parte, los videos con más de una etiqueta se consideran pertenecientes a todas esas clases simultáneamente. La figura 4.1 muestra un ejemplo de los casos anteiores.

(a) Video no perteneciente a alguna de las 20 clases, su etiqueta es 0.

(b) Video perteneciente a Music Performance, Wedding Reception y Wedding Dance.

Figura 4.1: Clase 21 y clases compuestas.

Se comenzó por agregar la clase 0 al análisis y se consideraron 21 clases en vez de 20. Esto forma parte de una decisión de implementación y diseño, ya que podría haberse eliminado esta nueva clase para calcular las métricas, pero manteniéndola en el modelo de entrenamiento, o bien, descartarla por completo. Se decidió de esta manera ya que en el set de testing existían consultas etiquetadas con 0, además, la cantidad de ejemplos que no pertenecen a ninguna clase, es aproximadamente un cuarto del total de videos, por lo que manteniéndola en las medidas de eficacia, aumenta el rendimiento del sistema.

A partir del punto anterior, se hizo visible el impacto que genera el diseño de CCV en las métricas de desempeño, ya que la elección de añadir ejemplos negativos a las 20 clases, influye en cómo debe ser medido el sistema, y obliga a escoger una configuración con respecto a la clase nueva. Esto significó, desde el punto de vista del trabajo realizado, adaptar la implementación para emular los resultados obtenidos por los autores del sistema original.

Siguiendo con la misma idea, el hecho de poseer videos con más de una etiqueta, significó una nueva decisión de implementación, ya que existe la alternativa de considerar al problema como multilabel, o tratarlo como clasificación habitual, pero al momento de calcular eficacia permitir cualquier clase del conjunto de etiquetas del video como correcta. La primera opción se implementó en base al mismo clasificador SVM, añadiendo, para cada combinación de múltiples clases en un video, una nueva etiqueta, es decir, todo video que estuviera catalogado con más de una alternativa, pasaba a formar parte de una nueva clase, correspondiente a todos aquellos videos que cumplen ese mismo grupo de etiquetas.

Un análisis de mAP, sobre esta manera de interpretar los datos, reveló que habían consultas que presentaban combinaciones de etiquetas que no estaban cubiertas en el set de entrenamiento, es decir, existían objetos de clases nuevas para el modelo, y los datos de training no podían brindar soporte para que dichas clases fueran reconocidas. Para ilustrar la situación anterior, sea E un conjunto de entrenamiento con videos pertenecientes a **Dog**, **Bird**, **Cat**, **Basketball** y **Beach**, además de eso, clases combinadas de ***Basketball+Beach***,

Dog+Cat y **Cat+Bird+Beach**, a partir de los cuales, el clasificador genera un modelo SVM capaz de predecir las 8 clases mencionadas. Sin embargo, sea T un conjunto de testing, que posee las clases **Dog**, **Bird**, **Cat**, **Basketball**, **Beach**, **Dog+Cat**, **Cat+Bird+Beach** y ***Dog+Cat+Bird***. Dado el conjunto T, y el modelo SVM obtenido, la clase combinada Dog+Cat+Bird es nueva para el modelo, y éste no podrá predecirla correctamente, por otra parte, la clase Basketball+Beach, que se encuentra cubierta en el modelo, no está representada en los datos de testing. Este ejemplo es similar a la situación real con el dataset completo, lo que llevó a pensar que CCV no fue diseñado para ser clasificado en un problema de multilabel. Por otra parte, realizando el análisis de la segunda estrategia, es decir, clasificación usual, y etiquetas opcionales, se llegó a resultados más cercanos de mAP con respecto a los declarados por los autores. De manera que, se cree que la forma como realizaron las mediciones publicadas en Jiang [2012] es empleando esta estrategia.

De lo anterior, se concluyó que existen diversas maneras de evaluar un sistema de clasificación, aun cuando se hayan decidido las métricas a utilizar, los datos pueden contar con información que debe ser interpretada, y como consecuencia se deben tomar decisiones que influyen en los valores finales de rendimiento. Además, cabe destacar que la forma en la que se diseña un dataset depende de un grupo humano, por lo que no existe una manera única de realizar este proceso, y cada decisión puede ser cuestionada, como es el caso del número de ejemplos pertenecientes a la clase 0.

4.2. Resultados de la experimentación

4.2.1. Evaluación del sistema Baseline

La siguiente tabla da cuenta de los resultados obtenidos para el sistema Baseline, es decir, con el módulo SVM como clasificador. Se especifican los valores de mAP para distintas combinaciones de descriptores, las que coinciden con las declaradas en el artículo original [Jiang, 2012]. La columna de mAP **implementado** corresponde a dicha métrica, calculada para el sistema que se desarrolló en esta memoria, por otra parte, la columna de mAP **original** contiene los resultados experimentales de los autores de SUPER. El objetivo de esta visualización es permitir una rápida comparación entre ambos sistemas.

Tabla 4.1: Comparación según mAP.

Descriptores	mAP(implementado)	mAP(original)
SIFT	0.494	0.523
STIP	0.351	0.449
MFCC	0.268	0.331
SIFT+STIP+MFCC	0.582	0.595
DURF (SURF)	0.352	0.513

Descriptores	mAP(implementado)	mAP(original)
CM	0.269	0.324
GIST	0.141	0.325
LBP	0.198	0.285
CM+GIST+LBP	0.306	0.438
Todos los descriptores	0.572 (sin SSD)	0.626 (con SSD)
MFCC + DURF	0.466	0.567
SIFT+STIP+MFCC+CM	0.591	-

Los resultados de mAP fueron utilizados para calibrar el sistema implementado, siendo considerado como prioritario el mAP de **SIFT+STIP+MFCC**, ya que esta combinación constituye el sistema base propuesto por los autores. Además, los histogramas de dichos descriptores se entregan junto con CCV, con lo que se evitaron las diferencias de implementación de ellos, y se asumió que no habían sido alterados. Como se puede apreciar, los valores son muy similares, con **0.582** para el sistema nuevo, y **0.595** para el original. En la sección de implementación del módulo SVM, se mencionó que este proceso de calibración tuvo varias etapas, y permitió concluir sobre la forma en la que se estaba midiendo la eficacia.

En cuanto a los tres descriptores por separado, se obtuvo la mayor diferencia para STIP, con 0.351 versus 0.449, es decir, aproximadamente 0.1. Dicho valor se consideró como un máximo aceptable para el resto de los descriptores y sus diferencias, por tanto, se cree que CM, MFCC, SIFT, LBP y STIP fueron correctamente implementados, y que los valores inferiores a los del sistema original se deben, principalmente, al módulo de clasificación. Cabe destacar que, a pesar de emplear kernels diferentes para SVM, se obtuvieron buenos resultados para histogramas de múltiples dimensiones. Sin embargo, los descriptores globales sufrieron una pérdida mayor en su rendimiento.

A partir de lo anterior, se puede ver que GIST y SURF son los únicos descriptores con un rendimiento más bajo de lo esperado. Llama la atención el caso de GIST, ya que su implementación corresponde a la misma que emplearon los autores del sistema original. Además, se corroboraron los parámetros utilizados para su extracción, y a pesar de realizar diversos experimentos, no se logró mejorar su mAP por sobre 0.141. Se cree que la pérdida de eficacia es debido al módulo de clasificación, pero se desconoce la fuente exacta del error, pudiendo ser ésta, la etapa de escalamiento requerida por PmSVM, o bien el uso de un kernel distinto.

Por parte de SURF, se esperaba un resultado cercano a los 0.494 de SIFT, ya que ambos descriptores son teóricamente similares. Sin embargo, su rendimiento inferior se explica por el proceso diferente al cual fue sometida su extracción, creación de vocabulario y cuantización. Por otra parte, lograr los 0.352 de SURF requirió modificar varias veces la implementación, ya que, en un principio, se obtuvieron valores cercanos a 0.1.

Otro foco de interés, es el hecho de que la baja eficacia de GIST y LBP los hizo disminuir los resultados del resto de descriptores al ser combinados, como por ejemplo, en la

configuración que cuenta con todos los tipos de características (a excepción de SSD, que no fue implementado), la que en su versión original alcanza un valor de 0.626 (con SSD), y que se cree que debiera haber entregado un valor entre dicho número y los 0.582 del sistema base (al no contar con SSD). Sin embargo, el valor de mAP fue de 0.572, lo que incluso es inferior al sistema base de SIFT+STIP+MFCC, es decir, GIST y LBP causaron una pérdida de eficacia, en vez de una mejoría. Cabe destacar que al combinar los tres descriptores con CM, sí se pudo apreciar una mejora, lo que se explica porque CM es el único de todos los descriptores que mide colores, por lo que su información es ortogonal al resto, y por tanto, es capaz de aportar nuevas características.

En general, se obtuvo un rendimiento menor al del sistema original, sin embargo, se consideran satisfactorios los resultados, ya que para los descriptores provistos (SIFT, STIP y MFCC), también es más baja su eficacia, por lo que se cree que los problemas se deben principalmente al clasificador, por sobre las implementaciones independientes de cada descriptor o sus extracciones.

Además del análisis de mAP, se experimentó con métricas basadas en las matrices de confusión obtenidas. Esto debido a que para KNN no se pudo calcular mAP, ya que para SVM se emplearon los puntajes del clasificador para crear un ranking, sin embargo, el uso de las distancias de KNN para formar el ranking, no depende del valor de K, por lo que para todo K se hubiera obtenido el mismo valor. Se planteó la posibilidad de incluir la variable K, no obstante, no se encontró una manera que permitiera puntuar a todo video utilizando K para ello.

Debido a lo anterior, se decidió trabajar con métricas empleadas en problemas de clasificación, y se cuestionó la validez de mAP, tanto para SVM como para KNN, pues se pensó que era una forma de mejorar los resultados de eficacia, y que no necesariamente estaba midiendo el correcto etiquetado de las consultas. Sobre este último punto, se descubrió que mAP tenía una alta correlación con las demás métricas, y por tanto, las dudas sobre su validez fueron despejadas. En la siguiente tabla se muestran algunos valores para el módulo SVM, con los que se pueden visualizar las correlaciones, específicamente, mAP, Overall Accuracy (Ov Acc), Average Accuracy (Av Acc) y Average Precision (AP).

Tabla 4.2: Correlación entre métricas.

Descriptores	mAP	Ov Acc	Av Acc	AP
SIFT	0.494	0.47	0.442	0.499
STIP	0.351	0.364	0.239	0.517
MFCC	0.268	0.325	0.226	0.335
SIFT+STIP+MFCC	0.582	0.5	0.396	0.679
DURF (SURF)	0.352	0.359	0.266	0.433
CM	0.269	0.319	0.193	0.38
GIST	0.141	0.226	0.124	0.156
LBP	0.198	0.262	0.162	0.216
Todos los descriptores	0.572	0.455	0.322	0.689

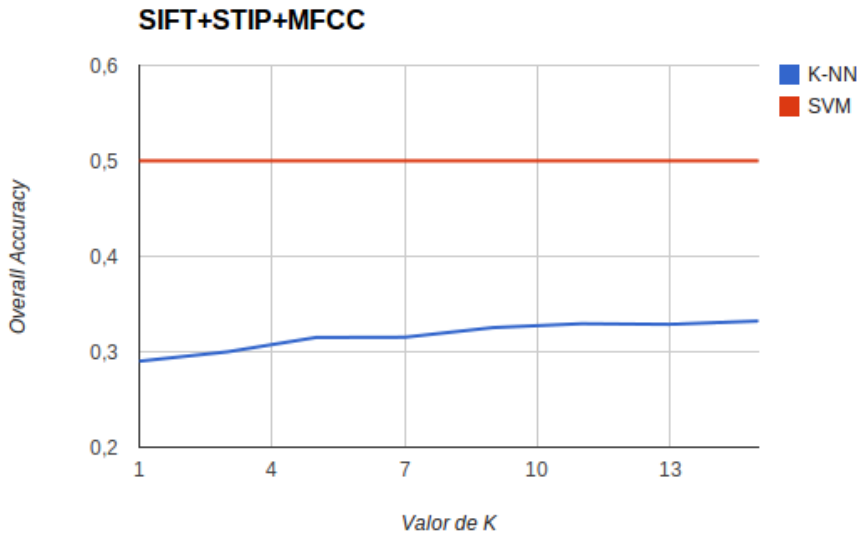
Descriptores	mAP	Ov Acc	Av Acc	AP
SIFT+STIP+MFCC+GIST	0.579	0.485	0.369	0.688
SIFT+STIP+MFCC+CM	0.591	0.483	0.365	0.716

4.2.2. Evaluación del sistema Modificado y comparación con Baseline

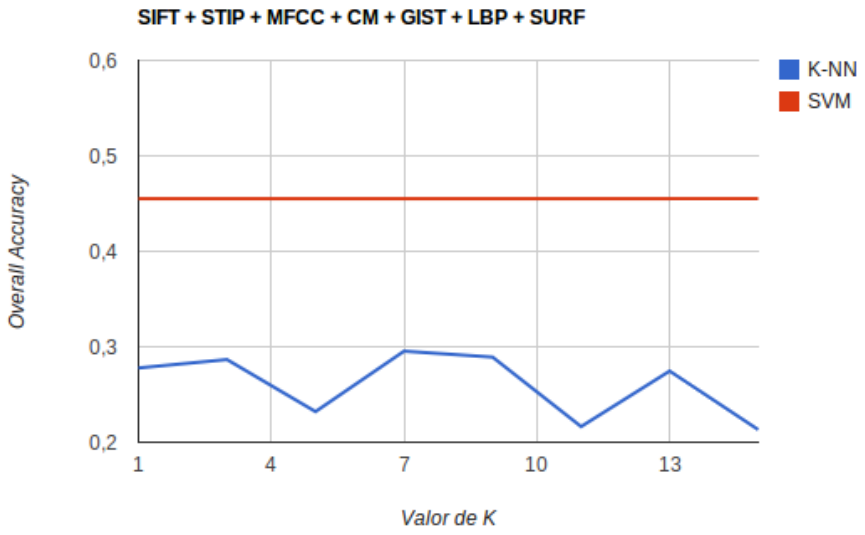
En esta sección se muestran los gráficos de las tres métricas basadas en matrices de confusión, realizando la comparación entre el sistema Baseline (SVM) y el sistema Modificado (K-NN). Debido al gran tiempo que requirió la utilización de KNN (en promedio 2 horas para cada valor de K), se escogieron sólo dos combinaciones de descriptores, la primera, SIFT+STIP+MFCC, y la segunda, el conjunto de todos los descriptores implementados. Los valores de K van desde 1 hasta 15, tomando únicamente los números impares.

A partir de los gráficos se observa que al añadir más descriptores se producen anomalías, por ejemplo, para la figura 4.2 existen caídas drásticas de Accuracy entre K=3 y K=5, y al llegar a K=9 y K=15 (considerando todos los descriptores). La explicación a esto se obtuvo visualizando la lista de vecinos de cada consulta, y para ilustrarlo, se considerará como ejemplo el video número 7 del set de testing, según el orden de los archivos provistos en la descarga de CCV (se mencionan los números de implementación de las clases en vez de sus nombres, en particular, la clase 0 corresponde a los ejemplos negativos o clase 21). La consulta de ejemplo pertenece a la clase 6, y las clases de sus cinco vecinos más cercanos, en orden creciente, son: 6, 0, 12, 9 y 0. Utilizando el algoritmo con K=5 se obtiene como etiqueta 0, sin embargo, con K=3, el resultado es 6 (se elige por defecto el primer vecino como correcto al haber empate). Este fenómeno se repite en otros videos, y se cree que la razón por la que ocurre es por la baja en el rendimiento producida por GIST y LBP, que genera listas con una menor tendencia hacia un valor (muchos valores aparecen una única vez), y donde cambios en K son los determinantes del resultado final. Además, es preciso recordar que la clase 0 posee un cuarto de videos de la colección, por lo que al aumentar K, es más probable que aparezca un objeto de dicha clase como nuevo vecino y las predicciones retornen 0.

Se puede notar que en todas las mediciones, el sistema Baseline supera ampliamente a Modificado, y que nuevamente, GIST y LBP disminuyeron la eficacia de los sistemas, resultando en un desempeño mucho menor para Modificado al utilizar todos los descriptores, en contraste con la combinación base de SIFT+STIP+MFCC. Se cree que este problema tiene relación con el kernel de SVM, ya que a pesar de ser chi cuadrado, y no exponencial de chi cuadrado, dicho kernel entrega mejores resultados para histogramas de múltiples dimensiones. A partir de lo anterior, se concluyó que el módulo de clasificación SVM es una mejor alternativa que KNN para los descriptores empleados, en el problema de reconocimiento de eventos en videos de consumidor.

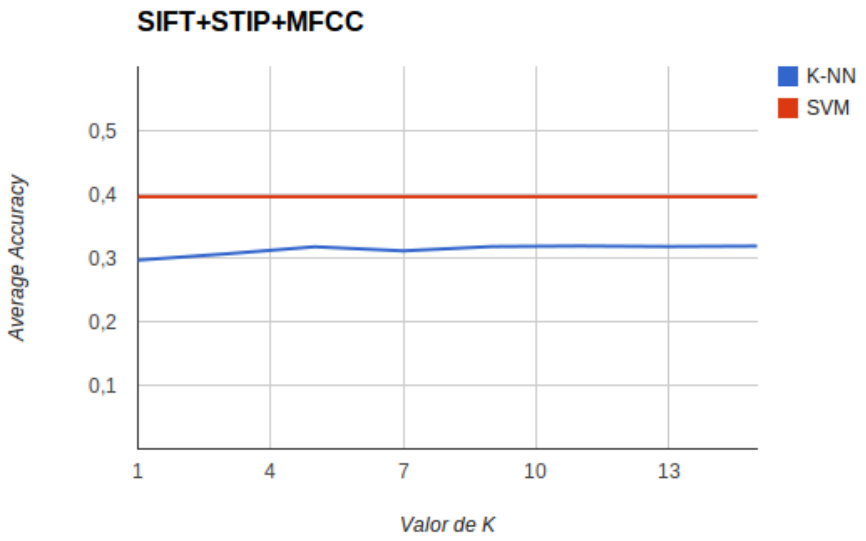


(a) Descriptores provistos

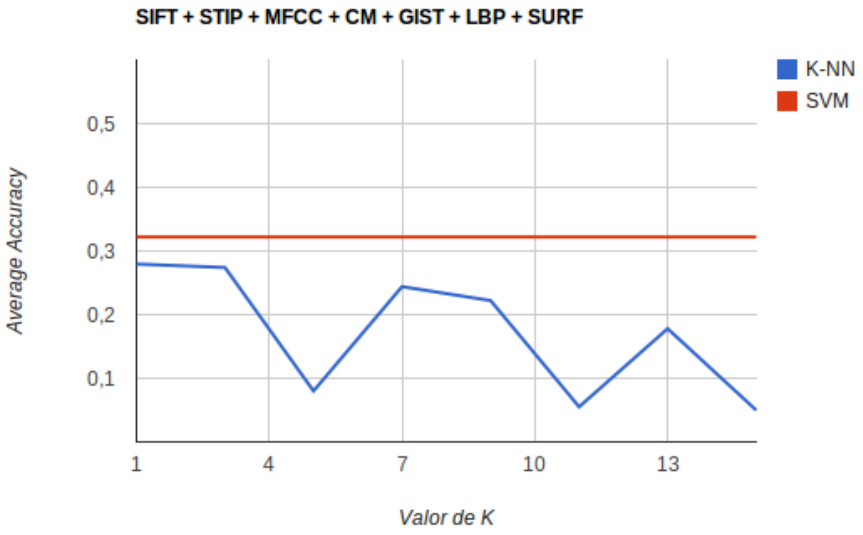


(b) Todos los descriptores implementados

Figura 4.2: Gráficos correspondientes a Overall Accuracy para ambos sistemas

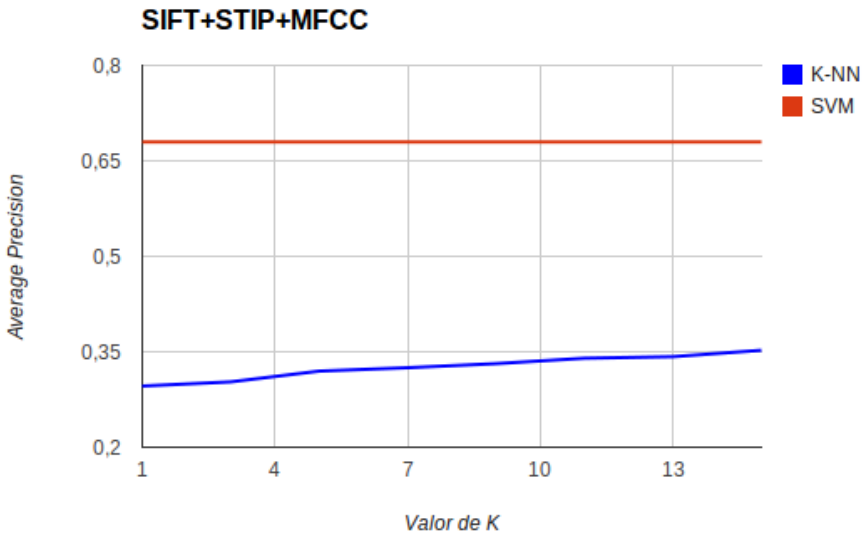


(a) Descriptores provistos

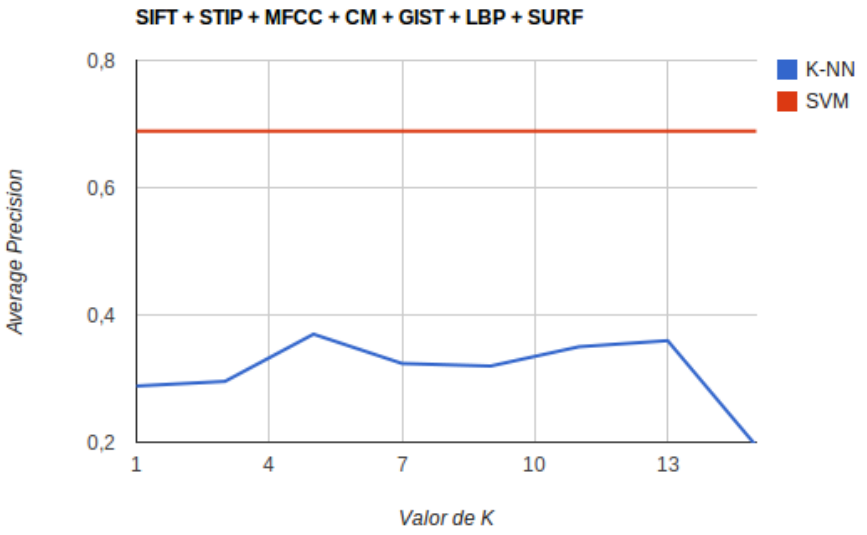


(b) Todos los descriptores implementados

Figura 4.3: Gráficos correspondientes a Average Accuracy para ambos sistemas



(a) Descriptores provistos



(b) Todos los descriptores implementados

Figura 4.4: Gráficos correspondientes a Average Precision para ambos sistemas

Capítulo 5

Conclusiones

Producto del trabajo realizado en esta memoria de título, se obtuvieron diversas conclusiones, las que se presentan a continuación.

En primer lugar, se debe destacar que existen descriptores cuyo tiempo de extracción dificulta que sean empleados sobre un volumen de datos como el dataset de CCV. Esto se debe a la naturaleza de los videos, entendidos como secuencias de imágenes, ya que el proceso implica procesar un conjunto de ellas, y condensar esa información en un único vector representativo del video. En particular, este problema surgió con SSD, y se descubrió la necesidad de un mayor poder de procesamiento, en caso de mantenerlo en el sistema. Los demás descriptores no presentaron dicho problema, a excepción de GIST, que tardó días en procesar la colección completa. Lo anterior muestra la importancia de una metodología para resolver el problema de procesar grandes volúmenes de videos, especialmente cuando el tiempo de cómputo para un sólo video es alto.

Continuando en el contexto de los descriptores, cabe destacar que las combinaciones que sean elegidas al momento de configurar un sistema como el reconocedor de eventos, determinarán qué tan eficaz resulta ser la clasificación. Esto pudo verse en la etapa de experimentación, ya que la combinación de SIFT, STIP y MFCC fue superada al considerar los mismos tres descriptores pero añadiendo CM en la evaluación, por otra parte, GIST y LBP redujeron el valor de mAP de SIFT, STIP y MFCC. Lo anterior da cuenta de la importancia de elegir las características que serán utilizadas para representar a los objetos multimedia, y plantea como posible extensión al sistema implementado, probar con descriptores que no hayan sido cubiertos en este trabajo.

En segundo lugar, en la etapa de creación de vocabularios nuevamente surgió el problema de procesar grandes cantidades de información, y no existía la posibilidad quitar del sistema dicha etapa, ya que eso hubiese implicado descartar los descriptores SURF también.

Para poder resolver el inconveniente se intentaron diversas opciones, todas ellas llevando al computador a sus límites de capacidad, y sin obtener resultados. Por tanto, se empleó una heurística de clusterización en dos fases y se redujeron los datos en un orden de magnitud. Este último paso permitió completar el proceso en un tiempo razonable, y dentro de lo permitido por el computador. La metodología utilizada no asegura una correctitud teórica, y puede haber introducido sesgo en los datos. Debido a lo anterior, además de reiterar la necesidad de mayor poder de procesamiento, plantea la posibilidad de desarrollar estrategias libres de sesgo, y utilizarlas para hacer frente al problema del gran volumen de datos, sin requerir equipos adicionales, o hardware más potente.

En tercer lugar, con respecto a la manera en la que se interpreta el problema de clasificación, se dejó en evidencia que al considerar una alternativa por sobre otra, los resultados de eficacia varían, e incluso, se pudo ver que CCV no estaba diseñada para utilizarse en un contexto multilabel. Esto hace notar que aunque se cuente con métricas de rendimiento, una parte importante del sistema está en la decisión de qué clases medir, como por ejemplo, si la clase adicional de consultas que no pertenecen a ninguna de las otras veinte debe ser parte de la evaluación o no. Esta información no estaba presente en el artículo original, y fue una decisión de implementación considerarla, ya que un cuarto de la colección correspondía a ese conjunto.

Por otra parte, hubo un proceso de entendimiento sobre mAP, ya que su origen proviene de sistemas de Information Retrieval, y se dudaba de su validez en un contexto de clasificación. Además, se concluyó que las múltiples etiquetas para un mismo video debían ser utilizadas como un conjunto, y dar la posibilidad de intercambiarlas en caso de mejorar mAP. Una explicación intuitiva al uso de esta métrica, y a un video perteneciente a más de una clase, consiste en interpretar el problema como una consulta por clases, y la respuesta como el ranking ordenado de videos. Por ejemplo, si la consulta es Basketball, se retornan todos los videos ordenados, e idealmente los primeros (con puntaje de clasificación mayor) son aquellos que el sistema estima como correspondientes a Basketball, en caso de que uno de ellos tenga múltiples etiquetas, si alguna de ellas coincide con Basketball, significa que el usuario que realiza la consulta recibe un video con más de un evento en él, pero que cubre la necesidad de búsqueda y debería ser relevante.

Siguiendo con mAP, su validez dejó de ser cuestionada al notar la correlación con las demás métricas. Sin embargo, se dejó como medición exclusiva para el sistema Baseline, ya que no fue posible formular una manera análoga de cálculo de mAP para el módulo K-NN del sistema Modificado, debido a la difícil inclusión de la variable K. En cuanto al módulo SVM, mAP fue empleado para comparar Baseline con el sistema original, y se cree que las diferencias (en algunos casos grandes) se deben a la implementación, especialmente al distinto tipo de kernel y al escalamiento de los histogramas. Se desconoce la razón exacta del bajo rendimiento alcanzado por GIST, ya que el descriptor fue implementado en base al mismo código utilizado por los autores, no obstante, se cree que el problema se relaciona con el clasificador.

Finalmente, en cuanto a los resultados de la comparación entre Baseline y Modificado, las métricas indicaron que el módulo SVM (a pesar de ser con un kernel distinto al original) fue superior en todos los experimentos, logrando niveles altos de eficacia. Se piensa que para los descriptores elegidos en el análisis, SVM es una mejor alternativa que K-NN, sin embargo, esto no implica que con otro tipo de descriptores, K-NN siga siendo inferior, incluso es posible que el dominio de los datos (consumer videos) haya influido en tan bajo rendimiento, y que si el sistema se probara con otro tipo de videos (o con ciertas modificaciones, sólo imágenes) el resultado podría ser diferente. Lo anterior indica que el resultado de la comparación entre SVM y K-NN no es generalizable a todo tipo de datos y configuraciones de descriptores.

Extensiones al trabajo realizado

A continuación se listan posibles extensiones a esta memoria, así como partes del sistema que no lograron ser implementadas y que sería deseable añadir.

- Calcular SSD, por ejemplo mediante un cluster de computadores, y añadirlo al sistema.
- Resolver la creación de vocabularios, ya sea a través de procesamiento, o de un algoritmo para enfrentar grandes volúmenes de información. Esta extensión implicaría una mayor cantidad de datos si se considera la opción de volver a incluir SSD, ya que, tanto SURF como SSD deben pasar por esta etapa.
- Probar el sistema con otro dominio de datos, por ejemplo videos de acciones humanas, y definir las clases que se medirán.
- Implementar SVM con el kernel exponencial de chi cuadrado y verificar mAP con respecto al sistema original. Medir GIST nuevamente, y verificar si en este nuevo contexto sigue entregando bajos resultados.
- Buscar un conjunto de descriptores que mejoren el rendimiento de K-NN, y compararlo con SVM. Aprovechar descriptores con características ortogonales entre sí, y medir su eficacia.

Apéndices

Apéndices A

Script de descarga de videos

```
#Se importa este módulo que permite crear pipes.
import subprocess

direccion = "http://www.youtube.com/watch?v="

#Se abre una estructura para el archivo que contiene las id, en
    este caso es de entrenamiento.
a = open("trainVidID.txt", "r")

#Se crea un archivo para recopilar los videos que no hayan sido
    encontrados o que su descarga se interrumpió.
b = open("logErrores.txt", "a")

#Cada línea es un id.
key = a.readline()

i = 1
while(key != ""):

        #youtube-dl espera una dirección en forma de url, por lo
            que se concatena la consante dirección con el id actual.
        video = direccion + key
        print "DESCARGANDO ", i

        #Se abre un proceso para youtube-dl y se conecta su salida
            a la salida de error.
        proc = subprocess.Popen(["youtube-dl", video], stderr=
            subprocess.PIPE)
```

```
        #Se busca en la salida de error la palabra ERROR que está
           asociada a videos no encontrados
        while True:
                line = proc.stderr.readline()
                if line != '':
                        if "ERROR" in line:
                                b.write( "ERROR: falta el numero "
                                    + str(i)+ " con id: "+key)
                                b.close()
                                b = open("logErrores.txt", "a")

                                break
                else:
                        break

        i += 1
        key = a.readline()
b.close()
a.close()

#Los videos descargados quedan automáticamente en el directorio
   donde se encuentra este ejecutable,
#Sus nombres corresponden al que poseen en youtube concatenado con
    el id de la variable key.
```

Apéndices B

Descriptor CM

```
#include <vector>

#include <cstdio>
#include <stdlib.h>

#include <iostream>

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/nonfree/nonfree.hpp"

using namespace cv;
using namespace std;

//Esta función calcula los valores de CM dada la imagen y un
    vector de double vacío

void calcularCM(Mat src, vector<double> * descriptor) {

  //Valores útiles de la imagen para determinar límites de
     iteración.
  int width = src.cols;
  int height = src.rows;
  int total = width*height;

  Mat lab;

  //Se convierte la imagen de entrada en RGB al espacio L*a*b
```

```
cvtColor(src,lab,CV_BGR2Lab);

//Se separa la imagen en sus tres canales
vector<Mat> canales;
split(lab, canales);

int i;

int x, y;

//Los tres momentos de CM.
double* primerCM = new double[3]();
double* segundoCM = new double[3]();
double* tercerCM = new double[3]();


//Se calcula el primer momento. (Promedio)
for (y = 0; y < height; ++y)
{
  uchar *filaL = (uchar*) ((canales[0]).data + (canales[0]).step
      * y);
  uchar *filaA = (uchar*) ((canales[1]).data + (canales[1]).step
      * y);
  uchar *filaB = (uchar*) ((canales[2]).data + (canales[2]).step
      * y);

  for (x = 0; x < width; ++x)
  {
    primerCM[0] += (double)filaB[x];
    primerCM[1] += (double)filaA[x];
    primerCM[2] += (double)filaL[x];
  }

}

for(i = 0; i < 3; i++)
{
  primerCM[i] = primerCM[i]/total;
  descriptor->push_back(primerCM[i]);
}

//Se calcula el segundo momento. (Desviación estándar)
//Se calcula el tercer momento. (Skewness)
for (y = 0; y < height; ++y)
{
  uchar *filaL = (uchar*) ((canales[0]).data + (canales[0]).step
      * y);
```

```
      uchar *filaA = (uchar*) ((canales[1]).data + (canales[1]).step
          * y);
      uchar *filaB = (uchar*) ((canales[2]).data + (canales[2]).step
          * y);

      for (x = 0; x < width; ++x)
      {

        double B = filaB[x];
        double A = filaA[x];
        double L = filaL[x];

        segundoCM[0] += pow((B - primerCM[0]), 2.0);
        segundoCM[1] += pow((A - primerCM[1]), 2.0);
        segundoCM[2] += pow((L - primerCM[2]), 2.0);

        tercerCM[0] += pow((B - primerCM[0]), 3.0);
        tercerCM[1] += pow((A - primerCM[1]), 3.0);
        tercerCM[2] += pow((L - primerCM[2]), 3.0);
      }

    }

    //Se normaliza el segundo momento.
    for(i = 0; i < 3; i++)
    {
      segundoCM[i] = segundoCM[i]/total;

      segundoCM[i] = sqrt(segundoCM[i]);

      descriptor->push_back(segundoCM[i]);
    }

    //Se normaliza el tercer momento.
    for(i = 0; i < 3; i++)
    {
      tercerCM[i] = tercerCM[i]/total;

      tercerCM[i] = cbrt(tercerCM[i]);

      descriptor->push_back(tercerCM[i]);
    }

    delete primerCM;
    delete segundoCM;
    delete tercerCM;
}
```

```
/*Esta función es la que se debe utilizar para calcular el
    descriptor CM
  simplifica la interfaz de la función anterior, y entrega un
     vector de 9 dimensiones.*/
vector<double> * CM(Mat src)
{

  vector<double> * descriptor = new vector<double>();

  calcularCM(src, descriptor);

  return descriptor;
}

/*Para calcular CM de 5 x 5 sobre la imagen, se requiere una
    función que divida
un objeto de tipo Mat en un vector de Mat de tales características
    ,
una implementación posible es la que se provee en el archivo de
    extracción de SURF
bajo el nombre dividir()*/
```

Apéndices C

Iterador de videos y extracción de SURF

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/nonfree/nonfree.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <cmath>
#include <vector>
#include <iostream>

#include <stdlib.h>
#include <cstdio>
#include <cstring>

using namespace cv;
using namespace std;

//Funcionalidad para leer un directorio y añadir los nombres de
   sus archivos en un vector de string.
int leer_directorio(string directorio, vector<string> *files){

        string dir = string(directorio);

        DIR *dp;
        struct dirent *dirp;

        string slash = "";

        unsigned int ultimo = dir.rfind("/");
        if (ultimo != string::npos && ultimo != (dir.length()-1))
```

```
        {
                slash = "/";
        }


        if((dp  = opendir(dir.c_str())) == NULL)
        {
                cout << "Error(" << errno << ")_opening_" << dir
                   << endl;
                return errno;
        }

        while ((dirp = readdir(dp)) != NULL)
        {
                string aux = string(dirp->d_name);
                if(aux.compare(".") != 0 && aux.compare("..") !=
                   0)
                {
                        stringstream st;
                        st << dir << slash << aux;
                        string staux = st.str();

                        files->push_back(staux);
                }
        }

        closedir(dp);

        return 0;
}

/*Esta función particiona una imagen en una malla de filas x
   columnas
se utiliza en el contexto de extracción de SURF, pero es
   perfectamente empleable
para los descriptores CM.*/

void dividir(Mat img, int filas, int columnas, vector<Mat> *
   particion)
{
    int width = img.cols;
    int height = img.rows;

    int stepX = width/columnas;
    int stepY = height/filas;

    int i;
```

```
    int veces_i = 0;
    for( i = 0; veces_i < columnas; i += stepX )
    {
        int j;
        int veces_j = 0;
        for( j = 0; veces_j < filas; j += stepY )
        {
            Mat tmp_img( img, Rect( i, j, stepX, stepY ) );

            particion->push_back(tmp_img);

            veces_j++;
        }

        veces_i++;
    }
}

//Retorna el tipo de archivo contenido en el string path.
string getExtension(string path)
{
    int punto = path.rfind(".");
    string extension = path.substr(punto + 1, path.length() -
       punto);
    return extension;
}

//Retorna el id del video contenido en el string path, se sabe que
    los id tienen largo 11.
string getID(string path)
{
    int punto = path.rfind(".");
    string id = path.substr(punto - 11, 11);
    return id;
}

/*Escribe en el archivo filename los datos de posición dentro de
   la imagen de un descriptor,
esto se basa en la información del KeyPoint kp, width y height
   corresponden a las medidas
del frame de video que se está procesando. Estos metadatos
   permiten calcular los descriptores SURF
una sola vez, y de esa forma, las divisiones espaciales de 2 x 2 y
    3 x 3, no requieren volver a extraerlos.
El objetivo es leer el archivo de metadatos y filtrar los
   descriptores contenidos en el archivo generado en el main.*/
```

```
void writeMetadatos(const char* filename, string id, int contador,
    KeyPoint kp, int width, int height)
{
    FILE *metadatosFile = fopen(filename, "a");

    int i22 = 2;
    int j22 = 2;

    int i31 = 3;
    int j31 = 1;

    if(kp.pt.y < height/2)
    {
        i22 = 1;
    }

    if(kp.pt.x < width/2)
    {
        j22 = 1;
    }

    if(kp.pt.y < height/3)
    {
        i31 = 1;
    }
    else
    {
        if(kp.pt.y < (2*height)/3)
        {
            i31 = 2;
        }
    }

    /*Un ejemplo de formato es: TzECsN0Y15o 1 22 31
    que significa que para el video con id TzECsN0Y15o, el
       correspondiente descriptor
    pertenece al frame 1, está ubicado en la malla de 2 x 2 en el
       cuadrante 2,2 y en la malla
    de 3 x 3 en la zona 3,1*/
    fprintf(metadatosFile, "%s %d %d%d %d%d\n", id.c_str(),
       contador, i22, j22, i31,j31);
    fclose(metadatosFile);
}

/*Esta función escribe en el archivo filename el contenido del
   descriptor SURF, cada línea
```

```
de ese archivo corresponde a un descriptor, y contiene el id del
    video, el número de frame
al que pertenece el descriptor, y su contenido de 64 dimensiones
    de float.*/
void writeDescriptor(const char* filename, string id, int contador
    , Mat descriptor)
{
    FILE *file = fopen(filename, "a");

    int width = descriptor.cols;

    fprintf(file, "%s %d ", id.c_str(), contador);

    int i;

    float *datos = (float*) (descriptor.data);

    for (i = 0; i < width; i++)
    {
        fprintf(file, "%f ", datos[i]);
    }

    fprintf(file, "\n");

    fclose(file);
}

/*Este método fue implementado en base a ejemplos de OpenCV
    encontrados en internet*/
int main(int argc, char* argv[])
{
    //Se debe entregar como parámetro el path del directorio que
        contenga todos los videos que serán procesados.
    const char* directorio = argv[1];

    //Se guarda en la variable archivos la lista de los path de
        todos los archivos presentes en el directorio.
    vector<string> *archivos = new vector<string>();
    leer_directorio(directorio, archivos);

    //Crea un detector de puntos de interés de SURF (algoritmo del
        Hessiano).
    SurfFeatureDetector detector(500, 4, 2, false, false);

    //Crea un extractor de características de tipo SURF.
    SurfDescriptorExtractor extractor;
```

```
unsigned int i;

for(i = 0; i < archivos->size(); i++)
{
    //Filtra archivos del directorio que no sean de tipo flv o
        mp4, ya que CCV contiene sólo estos formatos de video.
    string extension = getExtension(archivos->at(i));
    if(extension.compare(string("flv")) != 0 && extension.
       compare(string("mp4")) != 0)
    {
        continue;
    }

    //Se recupera el id del video.
    string id = getID(archivos->at(i));

    cout << "Procesando: " << id.c_str() << " -> " << i <<
       endl;

    //Abre el archivo de video.
    VideoCapture cap((archivos->at(i)).c_str());

    //Si no se puede abrir el video, se salta y se sigue con
       la lectura del resto.
    if ( !cap.isOpened() )
    {
        cout << "No se pudo abrir el archivo: " << (archivos->
           at(i)).c_str() << endl;
        continue;
    }

    //Valores útiles para determinar los frames que serán
       leídos (1 cada 2 segundos).
    double frames_per_second = cap.get(CV_CAP_PROP_FPS);
    int fps = (int)frames_per_second;
    double total_frames = cap.get(CV_CAP_PROP_FRAME_COUNT);


    int frame_actual = 0;
    Mat descriptors;
    int contador = 0;

    while(frame_actual < total_frames)
    {
        Mat frame;
        cap.set(CV_CAP_PROP_POS_FRAMES, frame_actual);
```

```
//Se lee un frame y se guarda en el Mat frame.
//Si no se puede leer el frame, se salta y se sigue el
    ciclo.
if (!cap.read(frame))
{
    cout << "No se pudo leer el frame en el archivo"
       << endl;
    frame_actual += 2*fps;
    continue;
}


//Se redimensiona la imagen a 320 x 240, en caso de
   tener un área mayor a 320*240
Mat dst;
if(frame.cols * frame.rows > 320*240)
{
    resize(frame, dst, Size(320, 240), 0, 0,
       INTER_LINEAR);
}
else
{
    dst = frame;
}


//Los puntos de interés se guardan en el vector
   keypoints.
vector<KeyPoint> keypoints;
detector.detect(dst, keypoints);

contador++;
cout << "frame: " << contador << endl;


int k;
for(k = 0; k < (int)keypoints.size(); k++)
{
    Mat descriptor;

    vector<KeyPoint> aux;
    KeyPoint kp = keypoints[k];
    aux.push_back(kp);

    extractor.compute(dst, aux, descriptor);
```

```
                /*Se escriben uno a uno los descriptores
                   detectados para el frame actual,
                para ello se cuenta con un nombre de archivo de
                   metadatos y otro para guardar las coordenadas,
                ambos archivos resultarán con la misma cantidad de
                    líneas, y existirá una correspondencia entre
                   ellas.
                Se deben entregar como parámetros los path de
                   dichos archivos.*/
                writeMetadatos(argv[3], id, contador, kp, dst.cols
                   , dst.rows);
                writeDescriptor(argv[2], id, contador, descriptor)
                   ;

            }

            //Se pasa al frame siguiente (1 cada 2 segundos).
            frame_actual += 2*fps;

        }

        //Se ha completado el proceso para un video.
        cout << id.c_str() << "␣Respaldado!!!" << endl;

    }

    return 0;

}
```

Apéndices D

Métricas para Baseline

A continuación se muestra una tabla con todas las métricas que se obtuvieron para el sistema Baseline en múltiples combinaciones de descriptores. Esto con el fin de entregar mayor transparencia para una eventual extensión a este trabajo de título, y servir como punto de comparación.

Descriptores	mAP	Ov Acc	Av Acc	AP
SIFT	0.494	0.47	0.442	0.499
STIP	0.351	0.364	0.239	0.517
MFCC	0.268	0.325	0.226	0.335
SIFT+STIP+MFCC	0.582	0.5	0.396	0.679
SURF	0.352	0.359	0.266	0.433
CM	0.269	0.319	0.193	0.38
GIST	0.141	0.226	0.124	0.156
LBP	0.198	0.262	0.162	0.216
Todos los descriptores	0.572	0.455	0.322	0.689
SIFT+STIP+MFCC+GIST	0.579	0.485	0.369	0.688
MFCC+SURF	0.466	0.418	0.304	0.582
SIFT+STIP+MFCC+LBP	0.579	0.488	0.376	0.705
SIFT+STIP+MFCC+CM	0.591	0.483	0.365	0.716
SIFT+STIP+MFCC+CM+LBP+GIST	0.569	0.459	0.329	0.723
SIFT+STIP+MFCC+CM+GIST	0.585	0.469	0.344	0.719
SIFT+STIP+MFCC+LBP+GIST	0.563	0.473	0.352	0.715
SIFT+STIP+MFCC+CM+GIST+SURF	0.581	0.463	0.334	0.728
SIFT+STIP+MFCC+CM+SURF	0.585	0.471	0.349	0.718
SIFT+STIP+MFCC+SURF	0.579	0.485	0.372	0.693
CM+LBP+GIST+SURF	0.387	0.361	0.224	0.0.446
MFCC+CM+LBP+GIST+SURF	0.479	0.389	0.243	0.651
MFCC+CM+GIST+SURF	0.496	0.389	0.247	0.688

Descriptores	mAP	Ov Acc	Av Acc	AP
MFCC+CM+SURF	0.503	0.402	0.267	0.697
MFCC+SIFT+CM	0.577	0.495	0.395	0.67
MFCC+STIP+SURF	0.519	0.426	0.294	0.687

Bibliografía

Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 020139829X.

Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008. ISSN 1077-3142. doi: 10.1016/j.cviu.2007.09.014. URL `http://dx.doi.org/10.1016/j.cviu.2007.09.014`.

K. Chatfield, J. Philbin, and A. Zisserman. Efficient retrieval of deformable shape classes using local self-similarities. In *Workshop on Non-rigid Shape Analysis and Deformable Image Alignment, ICCV*, 2009.

Yu-Gang Jiang. Super: towards real-time event recognition in internet videos. In *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*, ICMR '12, pages 7:1–7:8, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1329-2. doi: 10.1145/2324796.2324805. URL `http://doi.acm.org/10.1145/2324796.2324805`.

Yu-Gang Jiang, Guangnan Ye, Shih-Fu Chang, Daniel Ellis, and Alexander C. Loui. Consumer video understanding: a benchmark database and an evaluation of human and machine performance. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, ICMR '11, pages 29:1–29:8, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0336-1. doi: 10.1145/1991996.1992025. URL `http://doi.acm.org/10.1145/1991996.1992025`.

Yu-Gang Jiang, Qi Dai, Yingbin Zheng, Xiangyang Xue, Jie Liu, and Dong Wang. A fast video event recognition system and its application to video search. In *Proceedings of the 20th ACM international conference on Multimedia*, MM '12, pages 1347–1348, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1089-5. doi: 10.1145/2393347.2396477. URL `http://doi.acm.org/10.1145/2393347.2396477`.

Ivan Laptev. On space-time interest points. *Int. J. Comput. Vision*, 64(2-3):107–123, September 2005. ISSN 0920-5691. doi: 10.1007/s11263-005-1838-7. URL `http://dx.doi.org/10.1007/s11263-005-1838-7`.

Ivan Laptev, Marcin Marszalek, Cordelia Schmid, and Benjamin Rozenfeld. Learning Realistic Human Actions from Movies. In *IEEE Conference on Computer Vision & Pattern Recognition (CVPR '08)*, pages 1–8, Anchorage, United States, 2008. IEEE Computer Society. doi: 10.1109/CVPR.2008.4587756. URL `http://hal.inria.fr/inria-00548659`.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. URL `http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94`.

J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):971–987, July 2002. ISSN 0162-8828. doi: 10.1109/TPAMI.2002.1017623. URL `http://dx.doi.org/10.1109/TPAMI.2002.1017623`.

Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *Int. J. Comput. Vision*, 42(3):145–175, May 2001. ISSN 0920-5691. doi: 10.1023/A:1011139631724. URL `http://dx.doi.org/10.1023/A:1011139631724`.

Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local svm approach. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3 - Volume 03*, ICPR '04, pages 32–36, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2128-2. doi: 10.1109/ICPR.2004.747. URL `http://dx.doi.org/10.1109/ICPR.2004.747`.

Eli Shechtman and Michal Irani. Matching local self-similarities across images and videos. In *IEEE Conference on Computer Vision and Pattern Recognition 2007 (CVPR'07)*, June 2007.

Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1470–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1950-4. URL `http://dl.acm.org/citation.cfm?id=946247.946751`.

Markus Stricker and Markus Orengo. Similarity of color images. pages 381–392, 1995.

Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First*

Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321321367.

J. R.R. Uijlings, A. W.M. Smeulders, and R. J.H. Scha. Real-time visual concept classification. *Trans. Multi.*, 12(7):665–681, November 2010. ISSN 1520-9210. doi: 10.1109/TMM.2010.2052027. URL http://dx.doi.org/10.1109/TMM.2010.2052027.

Jianxin Wu. Power mean svm for large scale visual classification. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 2344–2351, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-1-4673-1226-4. URL http://dl.acm.org/citation.cfm?id=2354409.2354776.

Min Xu, Ling-Yu Duan, Jianfei Cai, Liang-Tien Chia, Changsheng Xu, and Qi Tian. Hmm-based audio keyword generation. In *Proceedings of the 5th Pacific Rim conference on Advances in Multimedia Information Processing - Volume Part III*, PCM'04, pages 566–574, Berlin, Heidelberg, 2004. Springer-Verlag. ISBN 3-540-23985-5, 978-3-540-23985-7. doi: 10.1007/978-3-540-30543-9_71. URL http://dx.doi.org/10.1007/978-3-540-30543-9_71.