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

CARTOON CHARACTER RECOGNITION: BÚSQUEDA Y RECONOCIMIENTO DE PERSONAJES ANIMADOS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

DANIEL ANDRÉS AVIV NOTARIO

PROFESOR GUÍA:
JUAN MANUEL BARRIOS NÚÑEZ

MIEMBROS DE LA COMISIÓN:
GONZALO NAVARRO BADINO
MAURICIO PALMA LIZANA

SANTIAGO DE CHILE
2016

CARTOON CHARACTER RECOGNITION: BÚSQUEDA Y RECONOCIMIENTO DE PERSONAJES ANIMADOS

El objetivo del trabajo aquí descrito es enfrentar el problema de detección y reconocimiento de personajes animados en la animación japonesa, también llamada animé. La resolución de este problema es importante por varias razones; no sólo pertenece a una familia de problemas difíciles de resolver, sino que solucionarlo significa la construcción de variadas herramientas para problemas reales en la vida diaria de animadores, editores y consumidores de animación japonesa en el mundo.

Para resolver el problema, se propone un proceso de solución compuesto por 4 subprocesos que cumplen tareas específicas: la determinación de fotogramas dentro del material animado, la detección de rostros dentro de los fotogramas seleccionados, el diseño y la determinación de características visuales que describan los rostros detectados y, por último, la determinación de medidas de comparación para las características visuales antes calculadas, y la posterior búsqueda de elementos similares dentro de un conjunto de datos.

En particular, la etapa de detección de rostros requiere el entrenamiento de un mecanismo de detección, lo que es costoso en términos de extracción de datos y tiempo de procesamiento. Además, la determinación de características visuales resulta difícil debido a la incertidumbre que la caracteriza.

En definitiva, el desafío abordado por este proyecto no sólo comprende implementar una solución que logre resolver los problemas planteados anteriormente, pero además lograr generar resultados que puedan competir con las mejores soluciones en la actualidad, ya sea en precisión o performance, y además procurando no descuidar las buenas prácticas de desarrollo como la inclusión de tests o documentación.

Para la implementación de la solución se propone un conjunto de scripts en el lenguaje Python, aprovechando la facilidad de desarrollo y las ventajas que provee la biblioteca OpenCV para el análisis y procesamiento de imágenes, videos y de datos relacionados.

Finalmente, se definen ciertos experimentos que permiten evaluar la efectividad de la solución propuesta. En esta sección se ve demostrado que el problema no sólo es posible de resolver, sino que varios resultados exceden lo esperado en términos de precisión y performance. En particular, se observa que el mecanismo de detección propuesto alcanza una precisión de más del 80 % a pesar de haber sido entrenado con solamente 800 ejemplos positivos, mientras que la característica diseñada para el reconocimiento alcanza una precisión promedio del 35 % para las consultas elegidas.

Para mi mamá, Rosario Notario*, por ser un símbolo de trabajo, integridad y esfuerzo.*

Agradecimientos

Agradezco a **Constanza Faez** por ayudarme a buscar fondos para el entrenamiento del detector.

Agradezco al súper equipo de taggeo de frames compuesto por **Fernando Morales**, **Natalia Hernández**, **César Arriagada**, **Jaime Sanz**, **Elizabeth Marchant** y **Óscar Guajardo** que sin su ayuda, la evaluación de calidad del proyecto hubiese sido mucho más trabajosa y los resultados hubiesen sido insignificantes.

Agradezco a **Francisca Aros** por enseñarme que existía *Lightshot*[1] haciendo mi vida más fácil.

Agradezco a **Jorge Bahamonde** por enseñarme a ocupar BibTex.

Agradezco a todos los personajes de animé que han dedicado su vida para combatir las fuerzas del mal, especialmente a los protagonistas de **JoJo no Kimyou na Bouken**.

Agradezco a la **cerveza**, mi compañera y amiga todos estos años de estudio.

[1] `https://app.prntscr.com/es/`

Tabla de Contenido

Índice de Ilustraciones

Introducción

Contexto

Últimamente ha habido una gran atención hacia el desarrollo de aplicaciones que hacen uso de técnicas de procesamiento de media y de visión por computador. Ya sean imágenes o videos, esta atención se ve motivada por la gran variedad de aplicaciones y procesos que se pueden automatizar usando técnicas de este estilo. Estas aplicaciones pueden ser observadas día a día; yendo desde el reconocimiento de caras para ingreso a un edificio a la traducción de texto en tiempo real, la tendencia indica que el desarrollo de tecnologías relacionadas a visión por computador seguirá creciendo con el tiempo.

Es interesante considerar la cantidad de problemas aún no resueltos que tiene esta área de desarrollo, razón por la cual es tan atractiva para investigadores de todo el mundo.

Un problema al que se han enfrentado en varias ocasiones los centros de investigación de esta área es el conocido problema de *Object Recognition* (Reconocimiento de Objetos). Este problema puede ser planteado de varias formas distintas, en este documento se define de la siguiente manera: Es el proceso por el cual se busca la aparición de un objeto dentro de un conjunto de datos que puede estar compuesto tanto por imágenes, videos, nubes de puntos, etc; dado como entrada al algoritmo una pequeña muestra del objeto a buscar.

Hay varias formas de enfrentarse al problema de Reconocimiento de Objetos. Generalmente, al buscar un objeto en medios del tipo visual (es decir videos o fotos) se hace uso de un *descriptor*, que se define como una métrica o medida que permita describir fehacientemente una característica específica de ese medio. Los descriptores vienen construidos de variadas formas y cada día se publican nuevos descriptores que hacen competencia con los ya existentes en precisión y performance.

Una particularización al problema Reconocimiento de Objetos es el problema de *Face Recognition* (Reconocimiento de Rostros), que se refiere al ejercicio de reconocer uno o más rostros de personas en una imagen o video. Este problema ya ha sido abordado en numerosas ocasiones por otros autores y centros de investigación alrededor del mundo, y los resultados alcanzados son realmente prometedores.

Ahora, se invita al lector a considerar las siguientes preguntas: ¿Qué pasaría si se sigue el mismo proceso de Reconocimiento de Rostros pero en vez de usar material con rostros humanos, se usa material animado?, ¿Entregarían los mismos buenos resultados que se tienen

en el problema original? Y en el caso de no hacerlo, ¿Cómo se podría modificar el proceso para solucionar el problema aplicado a material animado? Este problema, en este documento se le denomina *Cartoon Recognition*. Varios investigadores, como Chau [3] y Takayama [12], se han hecho estas mismas preguntas en el pasado, concluyendo de forma categórica que el uso de las técnicas para resolver el problema de *Face Recognition* original, aplicadas a *Cartoon Recognition* son infructuosas e insatisfactorias.

En este documento se define como "material animado" todos aquellos videos o fotografías de aquellos objetos que no pertenecen a la vida real, sino que han sido dibujados por un artista o animados por un estudio de animación. Estos se diferencian en varias cosas del material no animado; las fotos y videos no animados suelen tener zonas de relieve y los objetos suelen tener un gradiente suave, al contrario, las imágenes animadas suelen caracterizarse por tener grandes zonas de color constante con bordes marcados y abruptos.

Este trabajo se enfoca en un tipo de animación en particular: el animé (de la pronunciación japonesa de la palabra en inglés *animation*), que se define como toda animación de procedencia japonesa. Se muestran ejemplos de animación japonesa en la Fig 1, versus aquellas que no lo son en la Fig 2.

La misma problemática puede ser planteada para la animación en general, y el problema y el proceso de solución no varían. En este proyecto se trabaja solamente con animación de procedencia japonesa debido a que acotar el conjunto de datos permite que se compartan ciertas características comunes entre los datos, y así el ajuste de las herramientas que componen al proyecto produce mejores resultados finales.

Figura 1: Ejemplos de personajes obtenidos de animé

Figura 2: Ejemplos de personajes obtenidos de animación occidental

Relevancia del Problema

El animé es un género de animación que lleva muchísimos años en producción. Las primeras animaciones de procedencia japonesa vienen de principios del siglo XX, pero no sería hasta mediados de los 70 que el género se haya masificado. Hoy en día se percibe que cada año se producen más y más series y películas animadas de procedencia japonesa, y que éstas están inmersas ya no sólo en Japón, sino que pertenecen a la cultura pop occidental. Debido a la masificación de este género en occidente, el problema de *Cartoon Recognition*, aplicado a este género de animación, se convierte en un problema muy interesante.

Hay muchas posibles herramientas que se pueden basar en la aplicación de técnicas de detección y reconocimiento de rostros a personajes animados. Probablemente, una de las más importantes y relevantes es la *Búsqueda de Personajes*, que se refiere a un sistema que, dada una imagen de un personaje, puede buscar en una base de datos o en la Web fotos del mismo personaje. Este tipo de búsquedas (llamadas búsquedas de imagen por contenido o CBIR de sus siglas en inglés, *Content Based Image Retrieval*) aún no están presentes en algunos de los mayores motores de búsqueda (ej. Yahoo, Bing, etc.), y en los que sí están presentes (ej. Google) son búsquedas genéricas y no específicas a la búsqueda de personajes. Sin embargo, este tipo de búsquedas podría resultar útil para ilustradores, animadores o para titulares de los derechos de autor, debido a que podrían buscar e identificar uso inapropiado o no autorizado del material que poseen, afirmación compartida por Takayama en [12].

También se podrían detectar tendencias entre personajes, identificando si un rasgo en particular hace a un personaje más o menos popular. Otro tipo de aplicaciones incluye el indexado, de modo de ordenar una cantidad de fotos o videos por una característica específica de uno o más personajes. Ambas aplicaciones resultan útiles para los editores y consumidores de animé alrededor del mundo.

Descripción General de la Solución

A continuación, se presentan las alternativas de solución y el modelo de solución propuesto para resolver el problema. Se ahonda en ambos aspectos en los capítulos 1 y 3 respectivamente, por favor referirse a estos para más detalles.

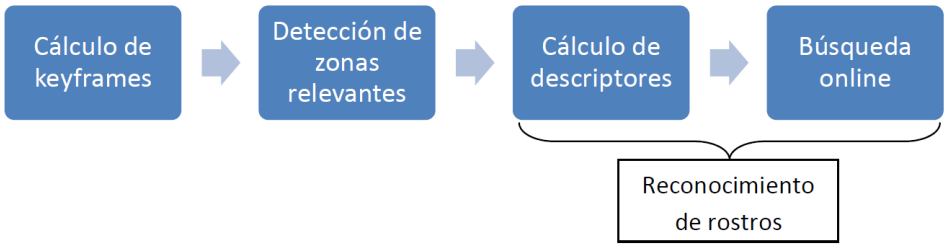

Figura 3: Diseño básico de la solución implementada

El proceso básico a seguir se indica en la Fig 3 y se detalla por pasos a continuación. La

descripción de estos procesos y la definición de los términos utilizados se hace con detalle en el Capítulo 1.

- **Determinación de Keyframes**
  Esta etapa es la primera dentro del proceso descrito y su objetivo es determinar qué fotogramas o *frames* del video se usarán posteriormente por el resto de las etapas, para ser resuelta hay dos alternativas básicas de solución para la determinación de keyframes, las cuales se explican con detalle en la Sección 1.2.

- **Detección de Rostros**
  La detección de rostros es sin duda una de las partes más importantes para asegurar el éxito del proyecto. El objetivo de esta fase es detectar rostros en los frames seleccionados por la etapa anterior. Se puede ahondar en estos conceptos en la Sección 1.3

- **Cálculo de Features**
  El cálculo de features es la siguiente parte del proceso. El objetivo de esta parte es diseñar una característica visual que describa bien las caras de animé (a esta característica o conjunto de características la llamamos descriptor o *feature*) y computar este descriptor para las caras detectadas por el detector en la fase pasada. Se puede ahondar en estos conceptos en la Sección 1.4.

- **Búsqueda y Votación**
  La búsqueda de vecinos y posterior votación es la última parte del proyecto. El objetivo de este proceso es, dado ciertas imágenes de input, encontrar las imágenes más parecidas en el conjunto de datos disponibles. Se puede ahondar en estos conceptos en la Sección 1.5. El output de este proceso representa el output final del proyecto.

Experimentos y Evaluación

A continuación se presentan los resultados de la solución propuesta. Se puede ahondar en estos resultados en el capítulo 4, por favor referirse a éste para más detalle.

- **Sobre la Evaluación del Proceso de Detección de Rostros**
  En esta etapa, las pruebas consisten en hacer iterar el detector en sus dos parámetros fundamentales y comparar estas iteraciones en términos de *precisión* y *recall*. Luego, se analiza cuál de éstas es la mejor configuración. Finalmente, se hace un análisis de performance al correr los tests. Dirigirse a la Sección 4.1 para más detalles.

- **Sobre la Evaluación del Proceso de Identificación de los Rostros**
  La evaluación de los features se hace en conjunto con la evaluación de la búsqueda. Las pruebas consisten en un análisis de precisión promedio para las consultas. Este proceso es realizado para dos versiones del feature, comparando el valor de ambas precisiones podemos seleccionar categóricamente al mejor. Dirigirse a la Sección 4.2 para más detalles.

Capítulo 1

Marco Teórico

En este capítulo se detallan las minucias técnicas del proyecto. Siguiendo el proceso planteado en la Fig 3, a cada uno de los puntos relevantes se le dedica una sección, pero primero se definen términos importantes para el entendimiento del capítulo.

1.1. Conceptos Importantes

En esta sección se definen los términos más relevantes para el correcto entendimiento del capítulo; puede que estos conceptos sean definidos nuevamente en la sección correspondiente.

- Dataset: Conjunto masivo de datos, habitualmente ordenados de alguna manera.

- Imagen: En este trabajo, definimos imagen como una función 2-dimensional *I(x, y)* donde x y y son llamadas coordenadas espaciales, estas coordenadas son valores enteros positivos limitados, $x \in [0, ancho - 1]$ e $y \in [0, alto - 1]$. Cada elemento definido por una posición se denomina *pixel*.

- Ventana: Se denomina ventana a un trozo o porción de una imagen, siendo más preciso, es una restricción del dominio de la función de la imagen original.

- Video: Conjunto de imágenes dispuestas y reproducidas una tras otra a alta velocidad con el propósito de generar la ilusión de movimiento para el receptor, usualmente, también son acompañadas por audio.

- Frame o Fotograma: Considerando que un video puede ser visto como un conjunto de imágenes reproducidas a cierta velocidad, llamamos *frame* o fotograma a cada una de esa imágenes que componen el video en su totalidad. Se hace la diferencia entre *frame* del término más general *imagen* en que un frame pertenece a un video.

- Keyframe: Frame clave, o de importancia para la ejecución de cierto algoritmo. Por

razones que se explican en la Sección 1.2, no todos los frames son relevantes para la ejecución del proyecto.

- Framerate: Tasa de velocidad a la que se reproducen los frames en un video. Para una tira de celuloide el estándar son 24 fps (frames por segundo), mientras que para un video en digital se utilizan 30 fps.

- Shot: Conjunto de frames contiguos que representan una toma de cámara sin interrupción. Los shots se caracterizan por que los frames contiguos varían muy poco.

- Feature, Descriptor o Característica: Llamamos *feature* a un característica específica de un medio, ya sea una imagen, video, audio, texto, metadata, etc. Esta característica puede ser de diversa índole y puede estar representada de muchas maneras, usualmente como un arreglo de números.

- Espacio o Modelo de Color: Un espacio de color es un modelo matemático que permite representar los colores del espectro visible de forma numérica. Así, el valor de los píxeles de una imagen es representado con un conjunto de números (típicamente 3) en algún espacio de color. Se profundiza este concepto en la Sección 1.4.

  En seguida se describen los dos espacios de color necesarios para la total comprensión del documento:
  - **Espacio de Color RGB:** Es el espacio de color más comúnmente utilizado. Su nombre viene de sus siglas en inglés, *Red, Green and Blue*. RGB es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la combinación uniforme de los tres colores de luz primarios (rojo, azul y verde). De esta manera, cada valor de pixel está representado por una tripleta que contiene el valor de cada uno de los 3 canales (es decir, $I(x, y) = (r, g, b)$). Frecuentemente, cada uno de los canales se representa con un entero binario positivo de 8 bits, provocando el valor de un pixel sea de 24 bits. Con esta profundidad de color es posible representar $(2^8)^3 = 16{,}777{,}216$ colores distintos. En la Fig 1.1 se puede observar una representación gráfica del espacio.
  - **Espacio de Color HSV:** HSV es un modelo de color basado en 3 componentes, *Hue* (Tono o Matiz), *Saturation* (Saturación) y *Value* (Valor). El matiz representa el color puro que queremos codificar, usualmente se codifica usando un ángulo entre 0 y 360. La saturación representa la "pureza" del color, entre más pequeño sea, el color parecerá más grisáceo y decolorado. Finalmente, el valor representa su cercanía al negro, entre más bajo sea más oscuro resultará el color. Se observa una representación gráfica del modelo en la Fig 1.2.
- Imagen en escala de gris: Usualmente, las imágenes se encuentran en color, lo que, como fue discutido anteriormente, significa que el valor de cada pixel corresponde a varios valores (típicamente 3), cada uno de estos valores es llamado *canal*. Una imagen en escala de grises, en cambio, sólo cuenta con un canal; es decir, el valor de cada pixel es un solo número, este representa el promedio ponderado de sus canales en el espacio de colores en que la imagen original esté codificada.

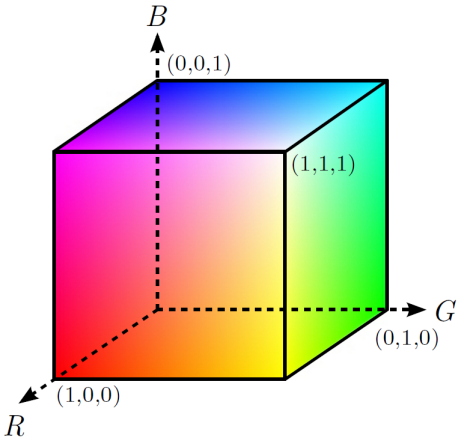

Figura 1.1: Representación gráfica del espacio de colores RGB.

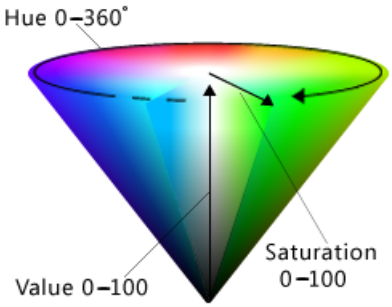

Figura 1.2: Representación gráfica del espacio de colores HSV.

- Histograma: En este documento, se define histograma como una medida estadística de frecuencia de eventos. Dadas n observaciones, los histogramas cuentan la cantidad de observaciones que caen en cada uno de sus k categorías disjuntas, a estas categorías las llamamos *bins*.

  Para una imagen, los histogramas sirven para hacer conteo del valor de pixeles y así sacar conclusiones de la distribución global de valores de una imagen, descartando información espacial. Considere una imagen en escala de grises donde el valor de cada pixel se codifican como un número entero entre 0 y 255, es decir, el espacio de valores posible es el intervalo $[0, 256]$, este se puede dividir, a modo de ejemplo, en 16 bins de la siguiente manera: $[0, 15] \cup [16, 31] \cup ...[224, 239] \cup [240, 255]$ y donde cada bin representa la cantidad de pixeles de la imagen original en su respectivo rango, así solamente se necesitarían 16 valores para representar la imagen.

  Los histogramas pueden ser visualizados para mejor comprensión, como se hace en la figura 1.3.

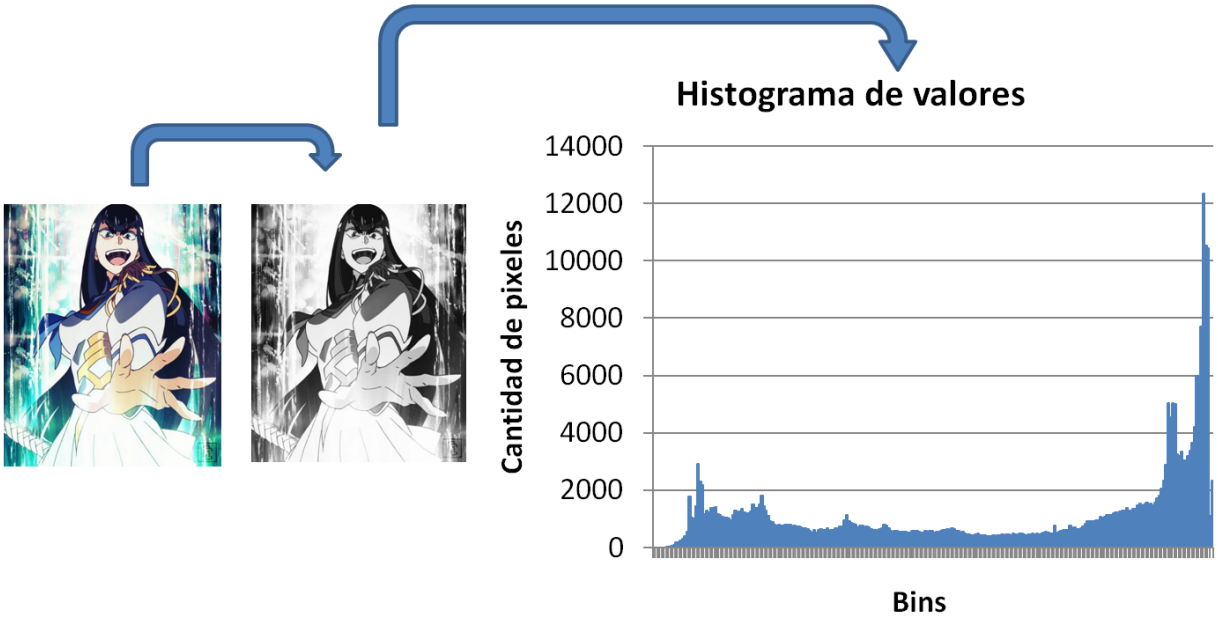

Figura 1.3: Ejemplo de histograma de grises.

1.2. Acerca del Cálculo de Keyframes

El primer punto relevante en el proceso de solución del problema es el cálculo de keyframes (fotogramas clave). El objetivo de este proceso es determinar ciertos fotogramas o *frames* del material animado con los que se trabaja.

No es recomendable (ni posible) trabajar con absolutamente todos los frames del video por varias razones. La primera, y la más importante, es que la cantidad de datos es inmanejable para un computador común y corriente; un capítulo normal de un animé dura aproximadamente 24 minutos y corre a un framerate de 30 *fps* (frames por segundo), esto significaría que para cada capítulo se deberían procesar 43200 frames. La segunda razón es que gran parte de los frames son redundantes, es decir, hay muchos frames consecutivos que tienen exactamente la misma información debido a que el cambio entre dos frames consecutivos suele ser muy bajo. En la Fig 1.4 se puede observar un ejemplo de lo anterior; el cambio entre (a) y (b) fue nulo, pero fue muy grande entre (b) y (c), lo que convierte a (b) en redundante.

Figura 1.4: Tres frames consecutivos obtenidos del animé *Gekkan Shoujo Nozaki-kun*.

Para resolver este problema hay dos alternativas de solución; la primera es utilizando un enfoque más directo, calculando un muestreo denso de todos los frames del video a tasa constante. Esto se refiere a trabajar con una tasa *"t"* constante de frames equidistantes. El valor de esta tasa es relevante, ya que entre más grande sea, se agrega mayor redundancia a los frames a analizar, sacrificando performance. La ventaja de este enfoque, además de su fácil implementación, es que con una tasa suficientemente alta (3-5 frame por segundos) no pierde información, es decir, muy probablemente no se pierda la aparición de un personaje importante. La desventaja es que de todos modos puede ser mucha información; a una tasa de 5 fps, en un capítulo de animé se trabaja con aproximadamente 7200 frames.

La alternativa al sampleo denso es hacer un preprocesamiento de los datos y hacer un cálculo de *shots* primero. Como se define en el glosario, un shot es una secuencia de frames en donde el cambio de frame a frame no es drástico, de esta manera un *shot* representa una escena entera sin cortes de cámara. Una vez calculados los shots, se extrae una tasa de frames constante de cada shot y esos frames son ocupados para el procesamiento, teniendo así la ventaja de que la cantidad de información se reduce drásticamente. La desventaja es que es un poco más complejo de implementar y puede que se pierda información relevante en algunos shots; esto último debido a que en un shot muchos eventos pueden pasar, incluyendo la aparición y desaparición de personajes, *panning* (barrido) de cámaras, zooms, etc.

Hay muchas formas distintas de calcular shots; Boreczky en [2] compara 5 distintas técnicas para material proveniente de películas, programas de televisión, noticiarios y comerciales. En este trabajo se concluye que los algoritmos más simples en general tienen mejores resultados que los más complejos. Algunos algoritmos que tuvieron buenos resultados son aquellos basados en *Region Histogram*, al igual que algoritmos del tipo *Running Histograms* similares a los descritos en [15].

1.3. Acerca de la Detección

El siguiente paso en el proceso para resolver el problema, es la etapa de detección. El objetivo de esta etapa es identificar cuáles de los frames determinados por la etapa anterior tienen rostros y dónde se encuentran aquellos rostros. Esta etapa es una de las más críticas para asegurar el éxito del proyecto.

Este problema ha sido enfrentado en el pasado por otros autores, en efecto existen algunas alternativas de detectores ya entrenados en la web. *OpenCV*[1] ofrece un entorno para hacer entrenamiento y detección, lamentablemente, los detectores de OpenCV fueron entrenados con caras humanas y no sirven para el contexto que se da el proyecto. También se encuentra Imager::AnimeFace[2], un detector que fue efectivamente entrenado con alrededor de aproximadamente 70.000 ejemplos de caras de animé y 300.000.000 ejemplos negativos, Chau en [3] calcula su precisión en alrededor del 61 %; sin embargo está implementado en Perl y Ruby, por lo tanto es incompatible con este proyecto.

[1] `http://opencv.org/`
[2] `http://anime.udp.jp/face-detect/`

Dado que ambas opciones resultan inadecuadas, se decide entrenar un detector propio. Para lograr lo anterior se pueden tomar dos enfoques: el primero es usar técnicas basadas en características (*feature-based face detection* o FBFD) y la segunda es usar técnicas basadas en imágenes (*image-based face detection* o IBFD).

FBFD propone calcular una característica visual específica de algún tipo (color, forma, dirección, etc.) sobre toda la imagen a detectar, luego de procesada esta característica se definen los candidatos del objeto a encontrar. Takayama en [12], para el problema de detección de rostros animados, propone primero filtrar todos los pixeles que pertenezcan a un espectro de color de piel (dejando de fuera los personajes con piel de color inusual como verde o azul), luego calcular bordes y separar por zonas, para finalmente eliminar las zonas que sean muy pequeñas o con dimensiones que no calcen con un rostro (sean muy alargadas o muy angostas), y las zonas que no cuenten con una simetría horizontal. Las zonas que pasen este proceso son consideradas candidatas a rostros.

La segunda alternativa es usar IBFD. Este enfoque es el propuesto por Paul Viola y Michael Jones en [13] y es un enfoque basado en aprendizaje computacional donde una función en cascada es ejecutada para la detección de un objeto habiendo sido entrenada previamente usando varios ejemplos positivos y negativos del objeto a detectar. En su caso general funciona usando características de tipo Haar, como las descritas por Papageorgiou et al. en [10], acelerando su cálculo usando imágenes integrales. De todas modos, no se pueden usar todos los features; con una ventana de entrenamiento de 24x24 se calcularían aproximadamente 180.000 features[3] que son, evidentemente, muchos más de los que se pueden manejar. Para seleccionar los mejores, se utiliza un algoritmo llamado **Adaboost** descrito por Freund en [4], el cual es discutido pero no profundizado en este documento.

Desde ahora se llamará detección sólo a las técnicas basadas en IBFD, ya que fueron las elegidas para desarrollar el proyecto.

1.3.1. Sobre el Mecanismo de Detección

En esta sección se explora el funcionamiento de los mecanismos y herramientas que hacen posible el entrenamiento.

Features Haar

Las features tipo Haar son descriptores visuales muy simples que usualmente son usados para reconocimiento de objetos.

Para calcular un feature Haar se define un rectángulo en alguna zona de la imagen, este rectángulo contiene zonas "blancas" y zonas "negras", el feature simplemente consiste en la diferencia entre la suma de los valores de los pixeles de las áreas blancas con la suma de los valores de los pixeles en las áreas negras, es decir:

[3] `http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html`

Definición 1.1 *Feature Haar*

$$H = \sum B(\mathrm{i}, j) - \sum N(\mathrm{i}, j).$$

Donde **B** son las zonas blancas y **N** las zonas negras. Este rectángulo definido por el feature puede tener cualquier tamaño, forma y estar en cualquier parte de la imagen. En la Fig 1.5 se muestran algunos ejemplos de features de tipo Haar.

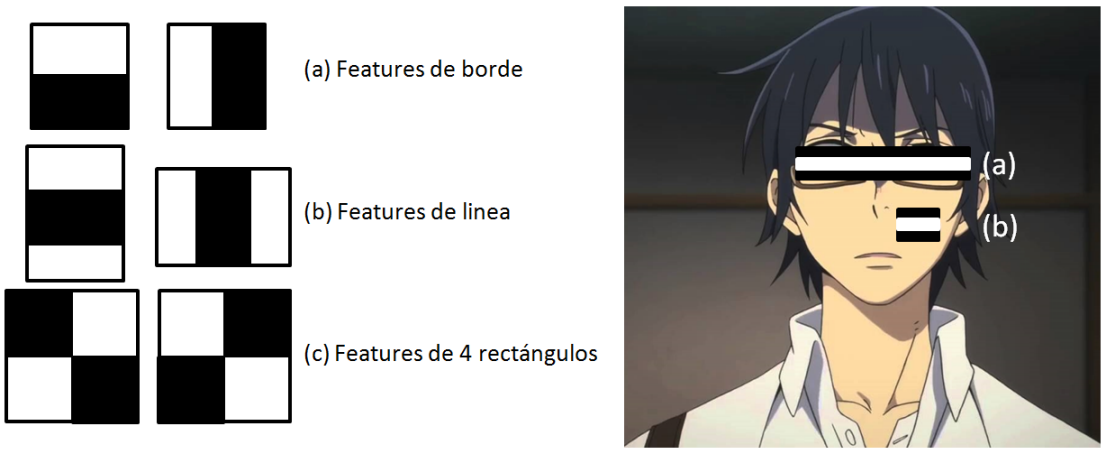

Figura 1.5: A la izquierda; ejemplos de features tipo Haar. A la derecha, aplicación de un feature a diversas zonas de una imagen.

Considerando sólo features Haar de 2, 3 y 4 rectángulos, y recordando que un feature puede tener un tamaño, forma y posición cualesquiera, nos damos cuenta que la cantidad de features posibles para un imagen es enorme, sin embargo, la calidad de estos features es variable; en la Fig 1.5, a la derecha, se puede ver la aplicación de dos features a una misma imagen, en este ejemplo podemos concluir categóricamente que el feature (a) es mejor que el feature (b); el feature (a) parece apoyarse en la propiedad de que la zona de los ojos varía en coloración, mientras que el feature (b) al estar ubicado en una zona plana como la mejilla es poco descriptivo y por lo tanto irrelevante.

Features LBP

Los feature LBP (de sus siglas en inglés *Local Binary Pattern*) son un tipo de descriptores visuales que, al igual que los features Haar, son usualmente usados para clasificación de imágenes. Fueron descritos por primera vez en 1994 por Ojala [9], y fueron usados en el contexto de clasificación en cascada en [7]. La ventaja de este algoritmo por sobre Haar es que es más veloz de calcular, sin embargo, la calidad dependerá del contexto en el que sea aplicado.
La forma de calcularlo es la siguiente:

```
LBP(V): #En donde V es una ventana en escala de grises.
```

1. Dividir la ventana V en celdas de tamaño constante (ejemplo: 16x16 pixeles).

2. Para cada celda C:
   (a) Para cada pixel P:
      - Comparar el valor de P con sus 8 vecinos (arriba, abajo, izquierda, derecha y en diagonal) en orden según las manecillas del reloj o en sentido contrario.
      - Para cada comparación, si el valor de P es mayor, concatenar un 0, de lo contrario, concatenar un 1.
      - El resultado es un número binario de 8 dígitos (o un decimal entre 0 y 255).
   (b) Construir un histograma de la frecuencia de aparición de cada numero entre 0 y 255 (creando un descriptor de 256 dimensiones).
   (c) Opcionalmente: Normalizar el histograma.
3. Concatenar los histogramas de todas las celdas.

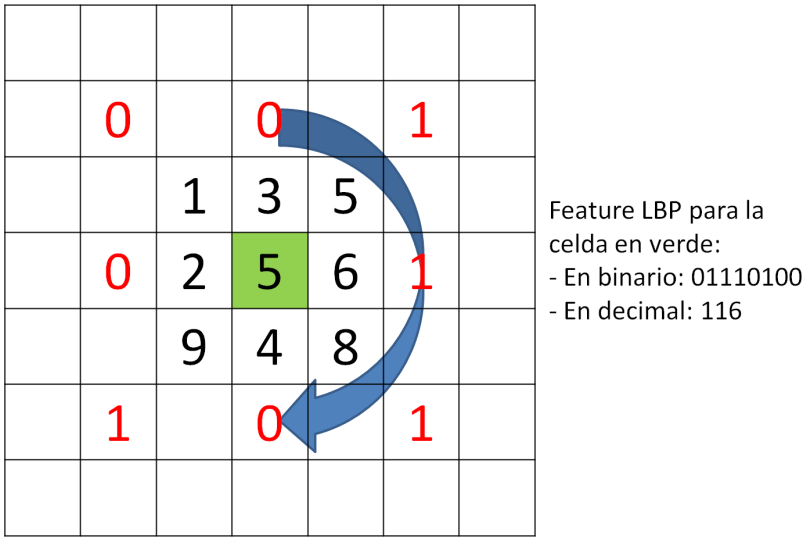

Figura 1.6: Ejemplo de cálculo de features de tipo LBP para un pixel.

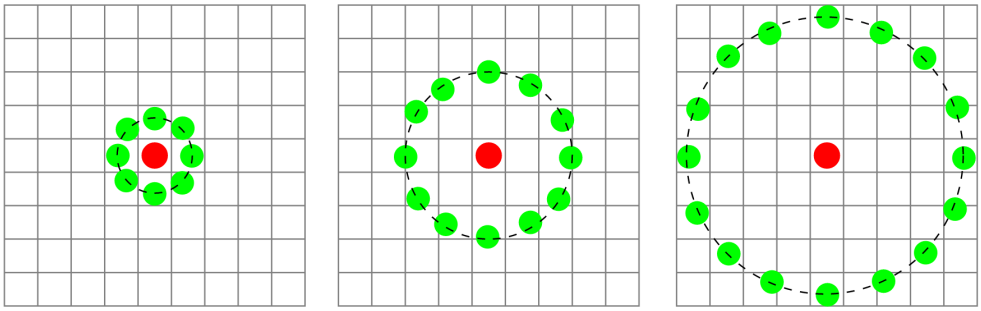

Figura 1.7: Ejemplos de vecindad posibles en el cálculo de features de tipo LBP.

Se puede observar un ejemplo del cálculo de LBP para un pixel en la figura 1.6, en esta imagen, se hace las comparaciones partiendo por el valor de arriba y siguiendo el sentido según las manecillas del reloj.

Terminado todo el proceso, el resultado es un feature de tamaño $256 \times cantidad\_de\_celdas$, donde la cantidad de celdas dependerá del tamaño de la ventana. Se puede probar distintas combinaciones de vecindad y no sólo la clásica, como se observa en la Fig 1.7.

Imagen Integral

Para asegurar una buena performance en el cálculo de features se hace uso de *imágenes integrales*, descritas por Viola y Jones por primera vez el año 2001 en [14], y luego con más detalle en el 2004 en [13].

Una imagen integral es una transformación de una imagen que facilita el cálculo de features rectangulares (como los features de Haar). La idea de una imagen integral es que en cada posición (i, j) en vez de contener el valor del pixel en esa posición se tiene la suma de todos los pixeles hacia arriba y hacia la izquierda del punto, es decir, considerando que el origen está en la esquina superior izquierda:

Definición 1.2 *Imagen Integral*

$$I'(\mathrm{i}, j) = \sum_{x=0}^{\mathrm{i}} \sum_{y=0}^{j} I(x, y).$$

Donde I' es la imagen integral e I es la imagen original.

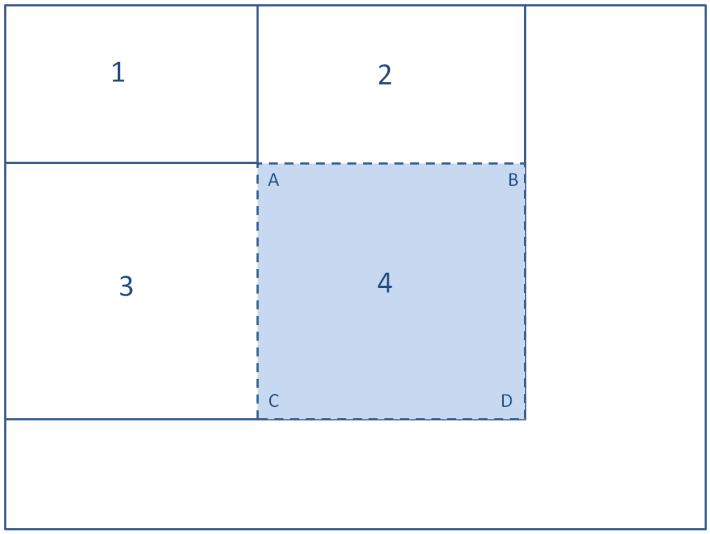

Figura 1.8: Cálculo de una imagen integral.

Usando imágenes integrales cualquier suma rectangular, para cualquier rectángulo y para cualquier escala, sólo requiere un operación que comprende el valor en sus 4 esquinas, en efecto, para un rectángulo cualquiera, si A es su esquina superior izquierda, B la esquina superior derecha, C la esquina inferior izquierda y D la esquina inferior derecha, la suma dentro del recuadro es: $D + A - (C + D)$, como se indica en la Fig 1.8.

Esta propiedad es fácilmente demostrable, si se observa la Fig 1.8 el punto A, tiene la suma acumulada de la zona (1), B la suma acumulada de (1) + (2), C contiene la suma acumulada de (1) + (3), y por último, el punto D contiene el equivalente de (1) + (2) + (3) + (4). Así, es fácil verificar que $D + A - (C + D)$ contiene la suma acumulada de la

zona (4). Esta propiedad hace a las imágenes integrales ideales para el cálculo de features rectangulares como los de Haar.

Habiendo entendido esto último es fácil verificar que se necesitan 6 referencias para features Haar de 2 rectángulos, 8 para los de 3 rectángulos y 9 para los de 4 rectángulos.

Discusión sobre Adaboost

Finalmente, y como se afirmó anteriormente, a pesar del uso de imágenes integrales, el cálculo de todos los features de Haar en una imagen es inviable, y considerando que, como se discute en el apartado de los features de Haar, varios features resultan irrelevantes, surge la necesidad de seleccionar aquellos más importantes.

Para eso se recurre a Adaboost, el cual es un algoritmo de selección de features. En términos generales, el funcionamiento del algoritmo se puede explicar de manera sencilla; para cada una de las imágenes de entrenamiento se aplican absolutamente todos los features, para cada feature se busca el mejor umbral de clasificación, evidentemente, en este punto van a haber errores de clasificación, en cada iteración nos vamos quedando con los features que tengan menor tasa de error, adicionalmente, a cada imagen se le asocia un peso, las imágenes que son clasificadas incorrectamente por algún feature aumenta de peso y se empieza una nueva iteración. El algoritmo sigue hasta que se alcanza una precisión o una tasa de error deseadas, o bien, hasta que se hayan eliminado suficientes features.

El clasificador final en la suma ponderada de todos estos features, los cuales se denominan *clasificadores débiles*. Se denominan de esa manera debido a que de forma independiente no pueden clasificar una imagen, pero en conjunto sí. El paper original de Viola [14] indica que para detección de rostros humanos alcanzaron una precisión de 95 % con 200 features, sin embargo su configuración final contenía 6000 features.

Definitivamente hay alternativas a Adaboost, Viola en [14] nombra alguna de estas, en particular menciona que Papageorgiou [10] propone un esquema de selección de features basado en varianza de features, demostrando buenos resultados logrando seleccionar 37 buenos features de 1734 posibles.

Sobre la Cascada de Clasificadores

Cuando llega el momento de clasificar, sin embargo, no se corren los 6000 features a todas las ventanas de la imagen (ya que resultaría extremadamente ineficiente). Para resolver esto, los autores introducen el concepto de **Clasificador en Cascada** o Cascada de Clasificadores. En vez de aplicar los 6000 features a todas las ventanas, se agrupan los features en distintas etapas las cuales se van aplicando de una en una.

Si una ventana falla la clasificación en una etapa (es decir, es considerada como un match negativo) no es considerada en etapas posteriores. Generalmente, las imágenes están compuestas por muchas zonas que no corresponden al objeto que se busca, considerando esto, las

primeras etapas suelen ser muy simples y contener un número pequeño de features de manera de descartar rápidamente la mayor cantidad de ventanas posibles. El proceso continua hasta que se hayan agotado todas las etapas.

1.3.2. Sobre el Entrenamiento

Para entrenar al detector se usa la aplicación de OpenCV *opencv_ traincascade*. Esta aplicación nos permite entrenar un detector en cascada usando features de tipo Haar o features de tipo LBP, que son más veloces que Haar.

Al hacer la fase de entrenamiento del detector es necesario considerar varios puntos importantes:

- La cantidad de imágenes: Generalmente, más es mejor. Para este proyecto se empezó con 500 ejemplos positivos y 300 negativos, y se observó que los resultados eran bastante subóptimos. Es por eso que se aumentó gradualmente el número de ejemplos y se llegó a 800 ejemplos positivos y 500 negativos. Hipotéticamente un orden mayor de ejemplos hubiesen mejorado el detector, pero recortar más imágenes escapaba del scope del proyecto.
- La calidad de las imágenes: Se observó un efecto interesante mientras se ejecutaba uno de los experimentos del proyecto. La cantidad de estudios que producen animé en Japón es enorme, y asimismo existen distintos estilos de animación; a pesar de que suelen converger y tener características o rasgos similares (ej. ojos grandes, nariz pequeña o inexistente, pelos de color no convencional, etc.) hay algunas series o películas que no siguen la convención (ver Fig 1.9). La adición de rostros de estilos no convencionales con los datos de entrenamiento hacía que el detector fuese más exhaustivo pero disminuía bastante la precisión.

Figura 1.9: Frames obtenidos de varios animés. La fila inferior representa animés de estilo clásico, la superior los que no lo son.

- El tamaño y dimensión de la ventana: Punto altamente relevante y bastante complejo, el tamaño de la ventana de entrenamiento afecta al resultado del detector de manera variable dependiendo del problema a resolver. Las dimensiones de la ventana tienen

que ser acorde al objeto a buscar, así, para este proyecto en relación con las caras de animé se utilizó una relación de aspecto de 91:100 (ancho:alto) aproximadamente.

- El algoritmo: La aplicación de *opencv_ traincascade* tiene la opción para entrenar el detector con features de tipo Haar o de tipo LBP, la elección del algoritmo es importante y afecta al resultado de la detección, esto sin mencionar que LBP tiene un tiempo de ejecución muchísimo menor al de los features de Haar.
- La cantidad de fases: La cantidad de fases de entrenamiento es un punto muy relevante a considerar, generalmente, entre más fases se entrene, más preciso y exhaustivo es el detector. Hay que tener consideración con el hecho de sobre-entrenarlo causando que el detector se vuelva *overfitted* y provocando un detector que no acepte ningún rostro del conjunto de test, es decir un detector con un recall muy bajo.

  Para entender porqué pasa esto, se ha proporcionado la Fig 1.10; en esta figura se representan dos espacios, uno bien entrenado y otro sobre-entrenado, los puntos azules representan ejemplos negativos, los puntos rojos ejemplos de entrenamiento positivos y la línea punteada representa el espacio vectorial que separa a ambos. Si se quisiese clasificar un nuevo dato (representado por el punto verde) cambiar de espacio cambiaría también el resultado. En el espacio bien entrenado, el punto verde calificaría como positivo, mientras que en el espacio sobre-entrenado no. Entre mayor cantidad de etapas lo entrenemos, más estricto será el espacio, provocando que finalmente no logre aceptar ningún dato que varíe de los datos de entrenamiento.

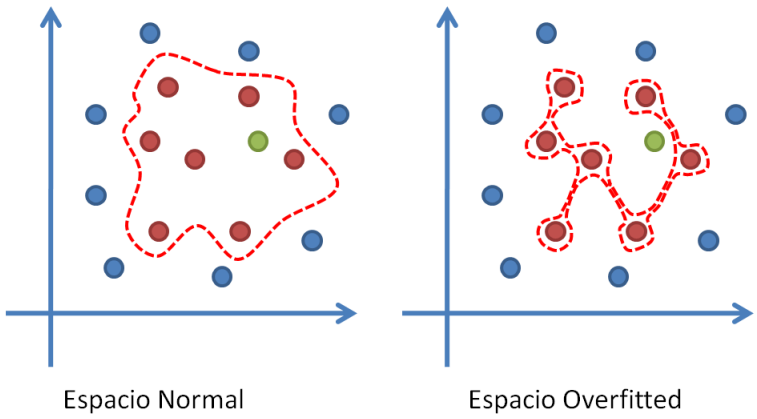

Figura 1.10: Diagrama que muestra como sobre-entrenar el detector puede ser contraproducente.

1.3.3. Sobre la Ejecución

La ejecución del detector es un proceso en el que juegan menos factores. Para ejecutar el detector que se entrenó, se ocupa la interfaz de OpenCV y, en su versión más básica, se requiere poner atención a dos factores:

- Factor de Escalamiento: Al hacer la detección, el modelo vectorial salido del entrenamiento tiene asociado un tamaño específico para detectar el objeto a buscar. Evidentemente, no todos los objetos van a tener el mismo tamaño en el conjunto de test, es

por eso que la imagen a testear se reescala varias veces, buscando el objeto target en cada uno de los escalamientos. Lo que indica el factor de escalamiento es con qué factor porcentual la imagen se reduce con cada paso del algoritmo. Entre más cercano a 1 sea, la cantidad de pasos aumenta, provocando que sea más factible encontrar un rostro, pero también aumentando el tiempo de ejecución considerablemente.

- Vecinos Mínimos: Cuando la ventana del detector barre sobre la imagen a buscar y encuentra un candidato a rostro, el detector corre la ventana un pixel esperando nuevamente encontrar el mismo objeto, este proceso se repite un cantidad de veces, si en todas estas ocasiones se encontró el objeto, este se considera un match positivo. La cantidad de veces está dada por el factor de *Mínimos Vecinos*, entre mayor sea el valor de este parámetro, más difícil será aceptar un candidato a match. En la Fig 1.11, podemos ver cómo afecta este valor a caras humanas, para este experimento se usó el detector de caras frontales que viene en OpenCV considerando un factor de escalamiento constante igual a 1.3, sin embargo se aumenta el factor de mínimos vecinos para los cuales se usan valores de 0, 3 y 5 respectivamente.

Cabe notar que la interfaz de OpenCV permite agregar más restricciones, pero sólo las dos nombradas anteriormente son consideradas obligatorias.

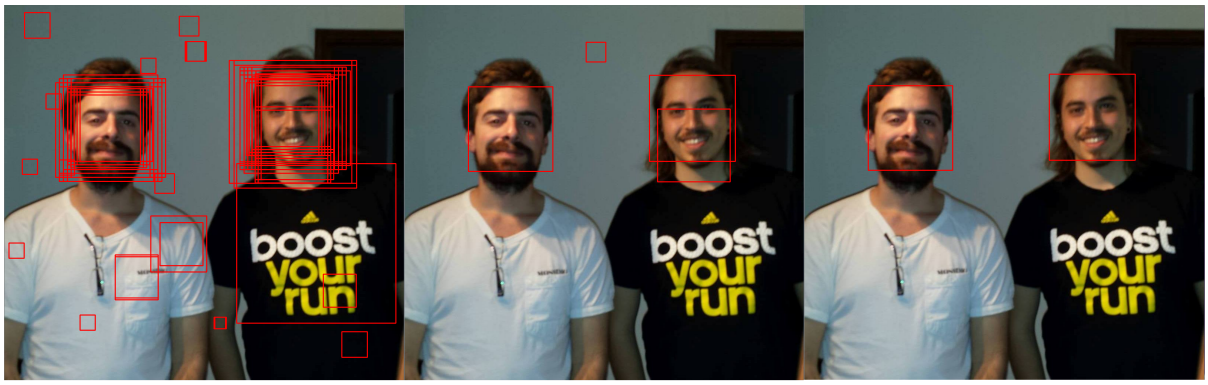

Figura 1.11: Detector de caras frontales de OpenCV con distinto valor de minNeighbours

1.4. Acerca del Cálculo de *Features*

El siguiente problema a resolver dentro del proceso es el cálculo de *features*. El objetivo de este proceso es calcular características visuales a los rostros entregados por el detector. A estas características visuales las llamaremos *descriptores* o *features*.

Los descriptores son de variada especie, los hay describiendo forma, color, textura, movimiento, orientación, etc. y su correcta elección para resolver un problema en específico puede resultar altamente no trivial.

Como se indica en la introducción de este documento, las técnicas para el reconocimiento de objetos reales resulta inadecuada para objetos animados; probablemente es el feature el responsable de lo anterior. La mayoría de aplicaciones de reconocimiento de objetos reales

Figura 1.12: Personajes del animé *Gekkan Shoujo Nozaki-kun*

se basa en descriptores de tipo SIFT como los descritos por Lowe en [8], los cuales están construidos a base de diferencia de gradientes. La razón por la que no funcionan para objetos animados es porque generalmente la animación carece de diferencia de gradiente, al contrario se ve caracterizada por grandes zonas de valor constante. Takayama en [12] menciona que usando métodos para gente real, los rostros animados son difícilmente detectados y menos aún reconocidos.

Para resolver el problema de la elección correcta de feature uno primero se debe preguntar qué caracteriza a un personaje animado. Otros autores han propuesto varias técnicas distintas, Arai en [1] menciona alguna de éstas. En particular menciona que Kawatani en [5] usa el contorno externo de la cara, el tamaño y forma del ojo y el color del pelo. Igata en [6], en cambio, propone la forma de la cara, tamaño del ojo y ancho de líneas para resolver el problema de diferenciación de sexo entre personajes de animé (es decir, identificar si son hombres o mujeres). En definitiva concluye que descriptores de pelo y ojo son los más adecuados para el reconocimiento de personajes de animé principalmente porque están presentes en casi todos los personajes (a diferencia de la nariz o las orejas, por ejemplo) y porque representan características que describen muy bien a un personaje.

En definitiva parece ser que los descriptores basados en color y forma son los mejores para abordar este problema, como se puede observar en la Fig 1.12, los colores del pelo suelen ser distintos dentro de los personajes de una serie, esto con el propósito de facilitar que el televidente sea capaz de reconocer al personaje de inmediato en diversas situaciones (de espalda, mirado desde arriba, agachado, etc.).

Para este proyecto se propone un descriptor basado en colores el cual se apodará Histograma de Tonos por zona o *HoH-zone* (de sus siglas en inglés Histograms of Hues by zone). Este descriptor es muy simple de implementar, pero muy poderoso para este tipo de problemas, además de ser relativamente rápido de calcular. Puede ser mejorado de muchas maneras sacrificando performance, tema que es discutido a continuación.

Para calcular HoH-zone primero se pasa la imagen objetivo del espacio de color RGB a

Figura 1.13: Esquema de la determinación de HoH-zone

HSV y se conserva solamente el primer canal, el de Tono o *Hue*, con el objetivo de hacerse indiferentes a cambios de iluminación; esto debido a que los personajes de animé pueden aparecer en escenas con luminosidad reducida (por ejemplo, una escena de noche). Una vez cambiado el canal, se divide la imagen en 4 zonas y en seguida se calcula 5 histogramas de intensidad, uno por cada zona y uno para la imagen entera. Los histogramas son calculados con 32 bins, haciendo que el detector en total sea de largo 160. Una esquema de HoH-zone se muestra en la Fig 1.13.

Se reconocen varias formas en la que el descriptor puede ser mejorado que, por limitaciones de tiempo del proyecto, no pudieron ser exploradas. Las opciones son realmente ilimitadas, pero con el propósito de completitud, algunas son nombradas a continuación. Una de las mejoras que fue considerada, era implementar un pre-filtro de pixeles; el propósito de esto es filtrar los candidatos a cara que no correspondiesen, para lograr esto se cuentan la cantidad de pixeles que pertenecen a un espectro de color de piel, este espacio de color es descrito por Tanvir en [11], si la cantidad de pixeles es suficientemente grande, al candidato se le calcula HoH-zone, de lo contrario, es descartado. Esta mejora no fue implementada ya que en etapas posteriores del proyecto se observó que el output del detector era muy bueno y no requería de este pre-filtro, el cual, cabe destacar, representa además un costo en performance. Otra posible mejora que se puede implementar al feature es una distribución de peso de los bins del histograma que represente mejor el problema; así para el problema que se enfrenta en este proyecto, se pesan más los pixeles de arriba de la imagen, debido a que los pixeles correspondientes al pelo son mucho más descriptivos que los de la piel.

1.5. Acerca de la Búsqueda

Por último y para terminar el proceso, debemos resolver el problema de votación. El objetivo de este proceso es, dadas ciertas imágenes de rostros de un personaje, determinar los $\boldsymbol{n}$ rostros del dataset que se parezcan más. Este proceso es un problema mucho más profundo y complicado de lo que uno pensaría y se deben considerar varias variables.

Debido a que para este trabajo la detección y el cálculo de features son más relevantes

para obtener buenos resultados en el reconocimiento de personajes animados, y considerando el tiempo limitado, no se evaluaron distintos mecanismos de votación y distancias.

El primer paso al hacer la votación es calcular los features (los mismos que fueron calculados al dataset) a las imágenes de input para luego comenzar la búsqueda. Este proceso se puede hacer de muchas formas, pero en este proyecto se ha hecho usando una búsqueda estilo kNN (de sus siglas en inglés, *k-Nearest Neighbours*, es decir los k vecinos más cercanos) para lo cual se usa la implementación de OpenCV[4].

kNN es un método para clasificar datos de manera supervisada. La idea del algoritmo es clasificar los datos al conjunto que contenga más miembros dentro de una vecindad, es decir, dado un dato a clasificar y para un k específico, se buscan los k vecinos más cercanos de ese dato en el dataset, el nuevo dato simplemente se agrega al conjunto que tenga más vecinos. Se puede hacer varias modificaciones al algoritmo original, por ejemplo, podemos considerar que no sólo la cantidad de miembros de cierto conjunto es relevante, sino que su distancia a ellos también debe ser considerada, así los vecinos más lejanos aportan menos a la clasificación que los que están más cerca.

La ejecución de kNN en el proyecto fue realizada de manera bastante similar al algoritmo original, se opera de la siguiente manera; para cada imagen de input se determinan sus $\boldsymbol{k}$ vecinos más cercanos en el dataset comparando los features de la imagen de input con los features del dataset, para hacer esta comparación la implementación de OpenCV usa Distancia Euclidiana[5]. Cada uno de esos k vecinos representa un voto de un input específico a un frame en particular del dataset, la forma más fácil de implementar esto, es sumar 1 por cada voto a un frame, pero, debido a que la distancia entre el primer y segundo voto puede ser muy grande, se decide implementar un sistema de votación con peso en las distancias, así, en vez de sumar 1 sumamos el inverso de la distancia entre el feature de input y el feature del dataset, esto con el propósito de que votos más lejanos (y por lo tanto menos significativos) aporten menos a la votación.

Una vez obtenidos los frames votados, simplemente se obtiene un número n de finalistas con mayor cantidad de votos y estos representan el output final del proyecto.

[4] `http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_ml/py_knn/py_knn_understanding/py_knn_understanding.html`

[5] Que es la distancia "ordinaria" entre dos puntos, es decir aquella medida con una regla.

Capítulo 2

Objetivos de Solución

En esta sección se discuten aquellos objetivos que debe ser capaz de cumplir la solución propuesta. También se establecen criterios de aceptación para estos objetivos.

2.1. Objetivo General

Como se indica previamente, en este trabajo se enfrenta el problema de Cartoon Character Detection and Recognition. El objetivo general de este trabajo consiste en poder generar una solución que sea efectivamente capaz de procesar los datos de la manera descrita en la sección anterior. Esta solución toma la forma de una aplicación de escritorio, de esta manera, esta aplicación debe ser capaz de hacer el cálculo de keyframes, la detección de rostros, el cálculo de features y la búsqueda de vecinos cercanos.

Sin embargo, no basta con sólo la construcción de la herramienta, también se plantea como objetivo la evaluación de ésta y su comparación con herramientas ya existentes.

Finalmente, se considera como objetivo secundario que esta aplicación sea rápida en procesamiento, fácil de usar y que cuente con test y documentación.

2.2. Objetivos Específicos

- Determinación del dataset: Primeramente, se requiere definir con qué datos se va a trabajar, la decisión de los datos evidentemente altera el output y la correctitud de la solución; esto se evidencia de varias maneras, por ejemplo, como se indica en la Sección 1.3.2 y se muestra en la Fig 1.9, distintos estilos de animación afectan el output del proyecto, también la longitud de los videos y el tamaño del dataset en total afecta el tiempo total de procesamiento de los datos.

- Dominio de acceso a los elementos del dataset: En varias partes del proyecto se requiere

acceder confiablemente a frames específicos de algún video, el caso más notable es al tener que calcular keyframes, ya que es el primer gran problema a resolver. Esto resulta ser una tarea para nada sencilla debido principalmente a cómo funcionan las librerías que manejan este tipo de funcionalidades y cómo reaccionan estas a ciertos tipos de encoding[1] de videos.

- Entrenamiento del detector: Similar al punto anterior, ganar dominio en las aplicaciones que hacen posible el entrenamiento del detector. Esto incluye la determinación del dataset de entrenamiento; tarea que si bien no es difícil, toma mucho tiempo. También incluye un proceso iterativo que permita entender el aporte de cada variable de entrenamiento a la resolución del problema específico.

- Entrenamiento del detector *exitoso*: El punto anterior se refiere al dominio del framework de entrenamiento, pero optimizar ese detector para que efectivamente obtenga buenos resultados es un problema distinto. Para este proceso se requiere iterar varias veces el detector sobre varios conjuntos de prueba distintos y verificar sus resultados.

- Diseño e implementación del feature: Luego de terminada la etapa de detección, el siguiente paso es el diseño del feature. La resolución de este problema requiere un profundo entendimiento de los factores elementales que componen el problema a resolver; así, se deben intuir ciertos factores que signifiquen mejores resultados, aquí entran en juego el tipo de feature, el tamaño de feature, la función de distancia, etc. Es, sin duda, la etapa con más especulación y la más susceptible a errores.

- Administración de datos: Este es un punto no menor del proyecto, ya que todos los módulos de alguna u otra forma deben procesar los datos del módulo anterior y generar los datos que sirven como input del módulo siguiente.

- Diseño e implementación de la búsqueda: Para finalizar el proceso se requiere hacer la búsqueda offline de los vecinos más cercanos entre los features de input y los del dataset. Este proceso es el más directo pero no el más sencillo, debido a ciertos detalles en la implementación.

- Usabilidad: Se debe contar con una interfaz de usuario que sea sencilla y fácil de ocupar.

- Documentación: Las clases deben contar con su respectiva documentación indicando cuál es la función a cumplir del módulo y los inputs y outputs de sus funciones.

- Tests: Se debe contar con una batería de tests para poner a prueba el correcto funcionamiento de los procesos en los casos que sean necesarios.

[1] Con *encoding* nos referimos al modo en que el video está comprimido.

Capítulo 3

Descripción de la Solución

En este capítulo se describen en detalle los pormenores de implementación del proyecto, tales como los requisitos de ejecución, la jerarquía de archivos, los detalles de implementación y el diseño de la interfaz de usuario.

3.1. Descripción General

La solución fue implementada en su totalidad en Python usando la versión 2.7.8. Los requisitos del sistema son los siguientes:

- `Python` en su versión 2.7.8 o compatible.[1]
- `NumPy` en su versión 1.9.1 o compatible.[2]
- `OpenCV` en su versión 2.4.9 o compatible.[3]
- El paquete `moviepy`.[4]

El entrenamiento del detector fue realizado usando las aplicaciones de OpenCV destinadas al entrenamiento de detectores, estas son `opencv_createsamples` y `opencv_traincascade`. Luego los detectores fueron calibrados experimentalmente.

Cada una de las etapas del proceso tiene un módulo dedicado para su ejecución. Además de estos, existen diversos otros módulos que sirven como tests o scripts de utilidad. No se utilizó ningún ambiente de programación en particular, sino que fue implementado mediante un editor de texto y ejecutado vía línea de comando. Todos los módulos principales y la mayoría de los módulos secundarios cuentan con su respectiva documentación.

Para realizar el manejo de los archivos se utilizó GitHub como repositorio en la siguiente dirección: `https://github.com/DanielAviv/CartoonRecognizer`.

[1] `https://www.python.org/`
[2] `http://www.numpy.org/`
[3] `http://opencv.org/`
[4] `http://zulko.github.io/moviepy/`

3.2. Metodología de Trabajo y Desarrollo

La metodología ocupada para desarrollar el proyecto, fue una basada en un modelo incremental apoyado por *Test-Driven Development*. La idea es generar resultados lo más rápido posible, para lo cual se desarrolló cada módulo siguiendo el orden de subprocesos descrito en la Fig 3 sin importar en primera instancia la calidad de los resultados.

Se tuvo que desarrollar los módulos siguiendo un orden lineal debido a que la salida de un módulo representaba la entrada de otro. Es debido a esta importante dependencia entre módulos que no se pudo desarrollar en paralelo. Cabe destacar que los módulos funcionan semi-independientemente (con una notable excepción) unos de otros, y pueden ser ocupados en conjunto o de manera *standalone*, con la única salvedad de mantener el formato de entrada correspondiente.

Cada módulo fue previa y posteriormente testeado usando la librería `unittest` de Python cuando lo necesitase. No todos los módulos fueron testeados previamente, debido a que gran parte de ellos no contaban con una interfaz para acceder a sus métodos individualmente, si no que más bien uno se comunica con los módulos mediante una interfaz por línea de comando y estos hacen su trabajo de manera que el usuario no se entera de la implementación.

Una vez terminado el desarrollo del proceso en totalidad, se retrocede en el proceso buscando falencias en el sistema y tratando de arreglarlas para mejorar la calidad del producto y la exactitud de cada uno de sus módulos.

El proceso descrito anteriormente se resume en la Fig 3.1.

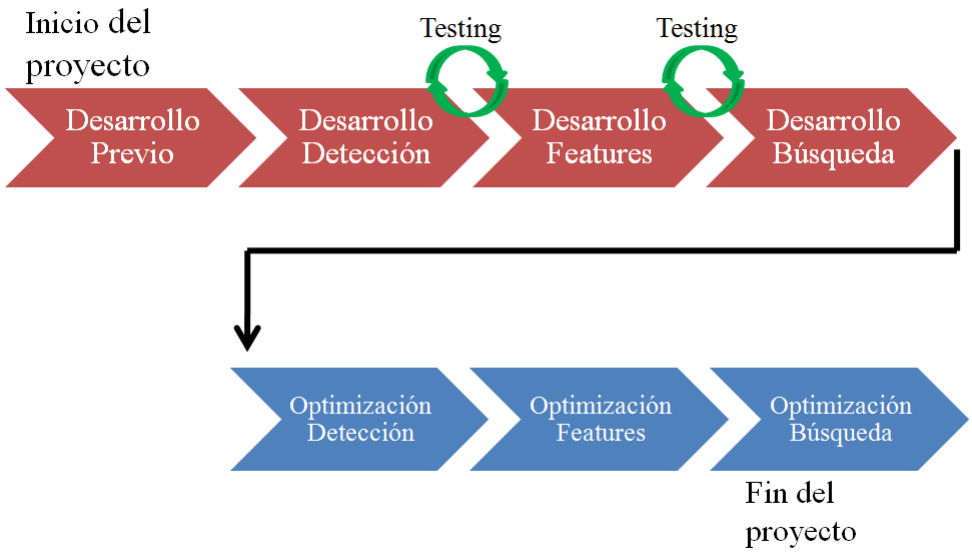

Figura 3.1: Proceso final de desarrollo.

3.3. Preprocesamiento

Previo al procesamiento se requiere cumplir ciertas condiciones. La primera, es la determinación del dataset. El dataset debe estar compuesto de videos, el largo y cantidad de los videos puede ser variable teniendo en consideración que a mayor cantidad de videos o a mayor sea el largo, mayor será el tiempo de procesamiento.

Para el proyecto se usan 100 horas de capítulos de diversas series de animación japonesa, cada video tiene una duración de alrededor de 24 minutos, teniendo un total de casi 300 videos los cuales son de una calidad de 720p o 1080p.

Lo segundo a considerar es el entrenamiento del detector. El paquete del proyecto contiene ya un detector entrenado y optimizado para el problema de búsqueda de rostros en el animé, pero si el usuario quisiese entrenar su propio detector, a continuación se detallan los pasos para hacerlo.

Primero, se requieren muchos ejemplos del objeto a detectar, entre más se tengan mejor será el detector. Para buscar un objeto sólido e inmutable como un logo, pocas imágenes serán suficientes, pero si se decide buscar caras, se necesitarán cientos o incluso miles de ejemplos. Para el entrenamiento del detector que definitivamente se ocupa en el proyecto se utilizan 800 ejemplos positivos.

Además de los ejemplos positivos, se requieren ejemplos negativos que son usados como background para el entrenamiento. Las imágenes negativas pueden ser cualquier y toda imagen que **no** contenga el objeto a detectar. Para el entrenamiento del detector del proyecto se utilizan 500 ejemplos negativos.

Realmente no hay una relación entre el número de imágenes positivas y negativas que sea "correcta", el éxito del proyecto va a depender del problema a resolver.

Una vez listas las imágenes se requiere generar dos archivos:

- El archivo positivo: Este es un archivo `.info` donde cada línea representa un ejemplo positivo. Cada línea tiene el path de la imagen seguido por la cantidad de objetos dentro de esa imagen, seguido por una serie de rectángulos indicando donde se encuentran los objetos dentro de la imagen.
- El archivo negativo: Este es un archivo `.txt` donde cada línea representa un ejemplo negativo. Cada línea es simplemente el path de la imagen.

Una vez generados los archivos se usa la aplicación de OpenCV `opencv_createsamples`, que se usa de la siguiente manera:

```
>opencv_createsamples.exe -info ARCHIVO_POS -num NUM_EJ_POS -w ANCHO
-h ALTO -vec OUTPUT
```

Los parámetros ANCHO y ALTO representan el tamaño de la ventana de entrenamiento, es muy importante usar una ventana que tenga una relación de aspecto similar al objeto a detectar, nuevamente el efecto del tamaño de la ventana va a depender del problema a

resolver. NUM_EJ_POS indica la cantidad de ejemplos positivos y OUTPUT es el path del archivo de output que debe ser con el formato `.vec`.

Ahora se utiliza la aplicación de OpenCV `opencv_traincascade`. Esta aplicación será la que finalmente entregue el detector para ser usado dentro del proyecto. Primero, se debe preparar una carpeta donde la aplicación pueda crear todos los archivos necesarios.

```
>opencv_traincascade.exe -data PATH_CARPETA_OUPUT -vec ARCHIVO_VEC
-bg ARCHIVO_NEG -numPos NUM_EJ_POS -numNeg NUM_EJ_NEG -numStages NUM_ETAPAS
-w ANCHO -h ALTO -featureType FEATURE
```

Debido al funcionamiento del algoritmo, NUM_EJ_POS debe ser **menor** al número real de ejemplos positivos. También hay que tener atención a que el largo y ancho de la ventana sean los mismos que fueron usados para el archivo `.vec`. Entre mayor sea el número de etapas, más preciso será el detector, teniendo el cuidado de no excederse causando que el detector quede *overfitted* (por motivos que fueron explicados en la Sección 1.3.2). Esta aplicación produce varios archivos, pero el llamado `cascade.xml` es el que usa para la detección.

3.4. Organización de los Módulos

En el siguiente diagrama se indica la organización de los módulos más importantes.

```
CartoonRecognition
├── ...
└── Source
    ├── data
    │   └── ...
    ├── test
    │   └── ...
    ├── util
    │   └── ...
    ├── create_info_file.py
    ├── detection.py
    ├── distance.py
    ├── feature_extraction.py
    ├── file_management.py
    └── precision_recall.py
```

3.5. Descripción de los Módulos Principales

`create_info_file.py`

Este módulo se encarga de crear los archivos necesarios para el entrenamiento del detector, es decir los que contienen los ejemplos positivos y negativos.

La interfaz de este módulo es mediante línea de comando. Sólo tiene un argumento opcional llamado `ws` que, de ser entregado, produce el archivo con los ejemplos negativos que contiene sólo las rutas de los archivos, en cambio, si no se entrega, produce el archivo con ejemplos positivos que contiene el path seguido de un "1" y seguido por el tamaño de la imagen. Asimismo, dada la descripción del módulo resulta evidente que sólo funciona cuando las imágenes usadas como ejemplos positivos de entrenamiento incluyen solamente al objeto a buscar (por ejemplo, para el problema de los rostros de animé, los ejemplos positivos son solo los rostros).

Para lograr lo anterior, el módulo simplemente itera sobre todas las imágenes, luego las abre usando OpenCV, extrae su tamaño y los escribe en el archivo de output.

`detection.py`

Este es el módulo encargado de hacer la detección de los rostros en el dataset.

La interfaz es vía línea de comando. Una de las opciones que se debe entregar al llamar al módulo es cuál detector se va a ocupar, las opciones válidas son `OCV` que ocupa el detector de rostros frontales de OpenCV y `DAN` que utiliza el detector entrenado y calibrado para rostros de animé en este proyecto. Si se desea utilizar otro detector, se debe cambiar la opción manualmente o reemplazar al detector en `CartoonRecognition/Source/data` (teniendo el cuidado de usar el mismo nombre para el archivo `.xml`). Luego de otorgado el detector, finalmente la interfaz pregunta en qué lugar se encuentran ubicados los videos.

Independiente del detector o de la ubicación del dataset se llama a la función `do_detect`. El objetivo de esta función es separar los videos del dataset en frames equidistantes y calcular el detector en los frames calculados. El output de esta función es un archivo por video conteniendo el nombre del video analizado, la posición relativa de los frames dentro del video que contengan el objeto a buscar y por cada frame un grupo de rectángulos indicando la posición del objeto detectado.

La implementación de `do_detect` puede parecer ineficiente, pero pruebas experimentales demuestran lo contrario. Éste simplemente recorre el video a detectar frame por frame deteniéndose regularmente para ejecutar el detector para ciertos frames. La frecuencia con la que se detiene la función está dada por una variable dentro del módulo, que por defecto es de 1 vez cada 10 frames, es decir, para un video fuente de 30 fps se calculará el detector para 3 fps. El output del detector se escribe en el archivo de output final.

`feature_extraction.py`

El módulo de feature extraction se encarga de calcular los features a partir del output del detector.

La interfaz de este módulo es vía línea de comando. El input necesario para el correcto funcionamiento de este módulo es la especificación de dos variables, el path donde se en-

cuentran los archivos de output del detector (que sirven como input del script), y el path de un directorio en donde el programa podrá generar su propio output. Este módulo llama a la función `calc_descriptors` que a su vez llama al descriptor elegido. Por ahora hay dos descriptores implementados en el módulo, pero solamente el descrito en la Sección 1.4, es decir HoH-zone, está disponible para su uso.

El output de este módulo es un archivo `.p` por video en el dataset. Cada uno de estos archivos contiene un diccionario donde la llave tiene codificado el nombre del video, el frame correspondiente y el rectángulo correspondiente a cada rostros en ese frame, el valor, en cambio, es el descriptor correspondiente a ese rectángulo.

Nuevamente, al igual que el módulo de `detection.py`, la implementación pareciese ser subóptima, pero resultados experimentales comprueban lo contrario. Para este módulo, nuevamente, sólo se recorren los frames del video secuencialmente buscando las posiciones descritas por el archivo input del script, una vez encontradas se recorta la cara y determina el feature correspondiente para luego guardar los resultados en el diccionario.

`file_management.py`

El propósito de este módulo es muy simple y es encargarse de la administración de los archivos dentro del proyecto. Simplemente sirve como interfaz para el paquete de Python `cPickle`. [5]

Contiene solamente 2 funciones; `save_file` recibe una estructura de datos serializable y una ruta, y produce un archivo `.p` en la ruta especificada con los datos entregados. En cambio, `load_file` recibe la ruta a un archivo `.p` y carga los datos dentro del archivo.

Este módulo es usado dentro del proyecto para guardar los descriptores calculados por el módulo `feature_extraction.py` y para cargarlos posteriormente en `distance.py`.

`distance.py`

Para finalizar con los módulos que pertenecen al proceso principal, se tiene el módulo de cálculo de distancias. El propósito de este módulo es, una vez calculado los features por el módulo de `feature_extraction.py` y dadas unas imágenes de input, buscar dentro del dataset las imágenes que más se parezcan al input.

Para interactuar con este módulo y al igual que módulos anteriores, se tiene una interfaz vía línea de comando. Para el correcto funcionamiento del módulo se requiere la especificación de 2 variables; el path del directorio en donde se encuentren los archivos generados por el módulo de `feature_extraction.py` y el path del directorio en donde se encuentren las imágenes de input.

El output de este módulo es un archivo `.txt` donde se especifican los **n** mejores candidatos

[5] `https://docs.python.org/2/library/pickle.html`

a match, el valor de **n** está especificado por una variable dentro del módulo y su default es 5.

Para la implementación de este módulo lo primero es calcular los features para las imágenes de input, una vez calculados se extraen los diccionarios de features de los archivos `.p`, con ambos features, los del input y los del dataset, se resuelve el problema de kNN para cada uno de los archivos de feature, para lo cual se recurre a la implementación de OpenCV. Una vez calculados los vecinos para cada archivo, se concatenan y se hace nuevamente un kNN del cual salen los candidatos a finalistas o "votantes".

Finalmente, y para terminar el proceso sigue la etapa de votación, aquí, cada votante le da un voto a un frame específico del dataset, siguiendo la estrategia indicada en la Sección 1.5.

`precision_recall.py`

Este módulo no pertenece realmente al proceso sino que cumple propósitos de calibración del detector. Su trabajo es observar el efecto de cambiar los parámetros `scale factor` y `minimum neighbours` en la precisión, exhaustividad y performance del detector y así poder elegir una configuración adecuada a la resolución del problema.

Se puede variar el rango y el step de las iteraciones tanto en `scale factor` como en `minimum neighbours` modificando el código. Como este módulo no pertenece al proceso regular, no tiene una interfaz para el usuario.

Capítulo 4

Experimentos y Evaluación

4.1. Evaluación del Proceso de Detección de Rostros

En esta sección se presentan los resultados del análisis del proceso de detección.

4.1.1. Conjunto de Prueba

Para hacer las pruebas se usa un conjunto de evaluación etiquetado *inhouse*, este consiste en aproximadamente 200 frames sacados de diversos animés en el dataset, de los cuales alrededor de la mitad tiene rostros mientras que el resto no los tiene.

Luego de determinados los frames de prueba, se etiqueta manualmente todos los rostros en los frames. Para hacer lo anterior, se dibuja un rectángulo alrededor del rostro en cuestión para así poder comparar con el resultado del detector. A este conjunto de rectángulos manualmente etiquetados se le llama *ground truth*.

Antes de presentar los resultados se deben hacer ciertas definiciones:

Definición 4.1 *Conceptos importantes*

- *Verdadero positivo: Se llama* verdadero positivo *a los rostros en el conjunto de test que el detector detecta correctamente.*
- *Falso positivo: Se llama* falso positivo *a todas las situaciones en que el detector indica que algo es un rostro sin serlo en el conjunto de prueba.*
- *Falso negativo: Se llama* falso negativo *a aquellos rostros en el dataset que el detector no logra detectar.*

Las pruebas presentadas a continuación se hicieron de la siguiente manera: por cada frame en el conjunto de prueba se ejecutó el detector y se comparó con el *ground truth*; si es que la intersección de un rectángulo del detector con un rectángulo del ground truth es superior al

50 % de la unión de ambos, este es considerado un verdadero positivo, si ningún rectángulo de los entregados por el detector cumple con la condición anterior para un rectángulo específico del conjunto de evaluación, se considera un falso negativo, asimismo, todo rectángulo del detector que no cumpla con la condición con algún rectángulo del conjunto de evaluación se considera un falso positivo.

Las pruebas se hicieron usando un mismo detector que fue entrenado inhouse usando 800 ejemplos positivos y 500 negativos recortados manualmente, los parámetros ocupados para el entrenamiento fueron los siguientes:

- Algoritmo: LBP.
- Etapas: 21.
- Tamaño de la Ventana: 55/50(alto/ancho).

Luego de entrenado el detector, fue iterado variando los valores de `Scale Factor`(SF) partiendo de 1.05 y terminando en 1.51 con un paso de 0.1, y los valores de `Minimum Neighbours`(MN) partiendo de 2 y terminando en 30 con un paso de 1, obteniendo así un total de 1363 iteraciones distintas del detector.

4.1.2. Precisión versus Recall

Para presentar los resultados a continuación es necesario primero hacer las siguientes definiciones:

Definición 4.2 *Conceptos importantes*

- *Precisión: En este contexto, se llama precisión al siguiente valor:*

$$P = \frac{verdaderos\_positivos}{verdaderos\_positivos + falsos\_positivos}.$$

- *Recall: Asimismo, se llama recall (o exhaustividad) al siguiente valor:*

$$R = \frac{verdaderos\_positivos}{positivos\_totales}.$$

Así, de manera coloquial, la precisión indica qué porcentaje de los resultados fueron realmente rostros, mientras que el recall indica el porcentaje de rostros encontrados por el detector dentro del total de rostros en los datos de test.

En el gráfico de la Fig 4.1, se ha graficado todas las iteraciones del detector indicando su precisión versus su recall. Como se puede observar en este gráfico, el detector tiene un comportamiento esperado siguiendo una relación inversa entre la precisión y el recall. También se observa que hay una envoltura convexa bien definida entre las iteraciones del detector, de esta manera sólo una cantidad pequeña de estas se mantiene competitiva.

Cabe destacar las siguientes tres iteraciones pertenecientes a la envoltura (marcadas en el gráfico con rombos rojos).

- (a) Scale Factor: 1.51, Min. Neighbours: 29 =>Recall: 19.3 %, Precisión: 100 %.
- (b) Scale Factor: 1.29, Min. Neighbours: 17 =>Recall: 73.76 %, Precisión: 84.65 %.
- (c) Scale Factor: 1.1, Min. Neighbours: 2 =>Recall: 94.55 %, Precisión: 10.58 %.

De estos valores es interesante notar la gran varianza que tienen los datos al modificar el valor de los parámetros para el mismo detector. El punto (a) tiene una precisión del 100 % para los datos de test, pero un recall muy bajo, provocando que detecte pocos rostros, pero los rostros detectados sean efectivamente verdaderos positivos con gran confianza, en cambio la configuración (c) detecta la gran mayoría de los rostros pero sólo un décimo de los rostros detectados son verdaderos positivos, finalmente se tiene la configuración (b) para la cual se tienen resultados equilibrados entre recall y precisión.[1]

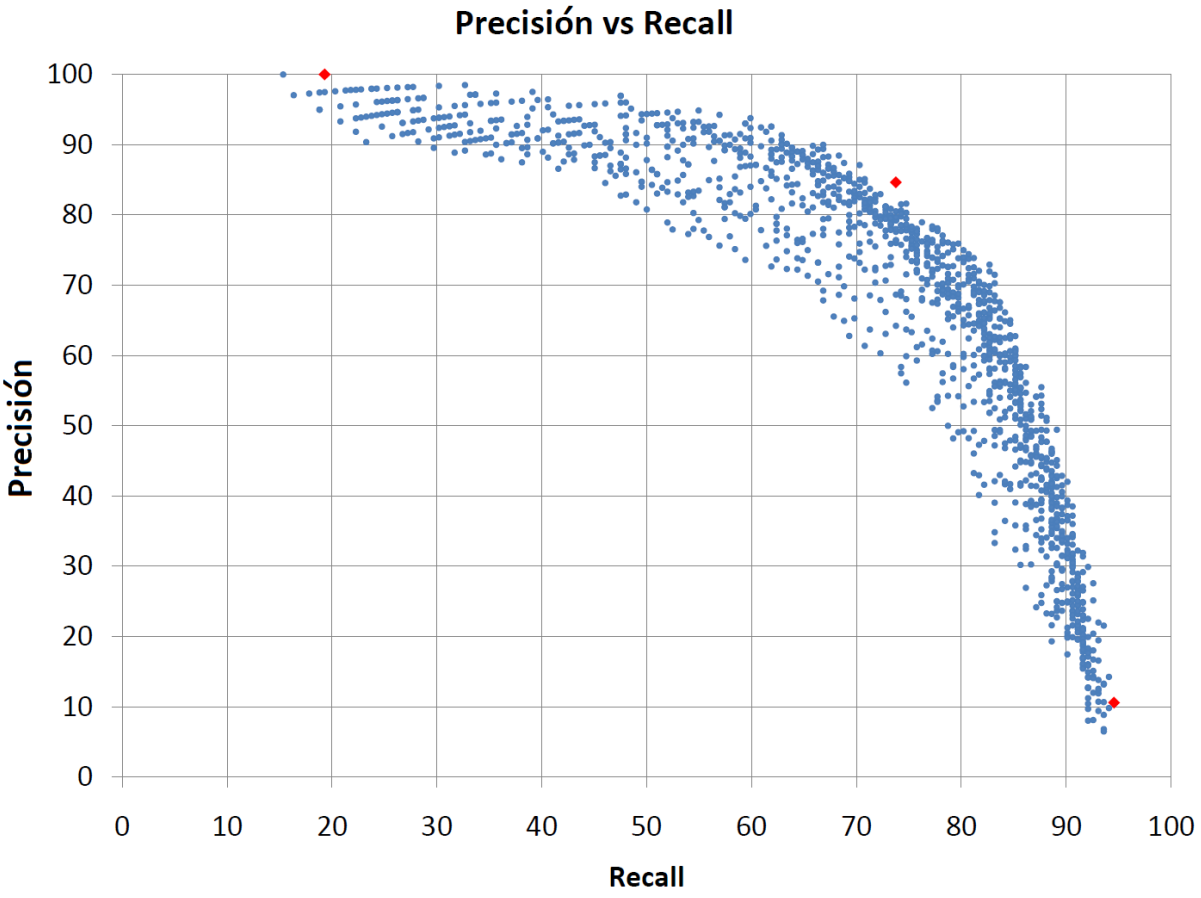

Figura 4.1: Gráfico precisión versus recall

Dado estos resultados es pertinente preguntar cuál de las configuraciones es la mejor, sin embargo no hay una respuesta única a esta pregunta ya que la respuesta va a depender del problema a resolver. Es por eso que para evaluar la calidad de las configuraciones ocupamos dos medidas distintas; la primera es la media aritmética (designada por la letra M), esta medida es útil pero puede que no represente la calidad real de la configuración debido a que, en general, además de querer una media alta queremos que los valores de recall y precisión

[1] Cabe notar que puede que estos resultados no se mantengan si son evaluados usando un distinto conjunto de prueba.

sean independientemente altos, es debido a eso que recurrimos a la segunda métrica, la media armónica (designada por la letra H) que se define a continuación como:

Definición 4.3 *Media armónica*

$$H = \frac{n}{\sum_{\mathrm{i}=1}^{n} \frac{1}{x_{\mathrm{i}}}} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + ... + \frac{1}{x_{n-1}} + \frac{1}{x_n}}.$$

En las tablas 4.1 y 4.2 se muestran las mejores cinco configuraciones para cada una de las métricas:[2]

Media Aritmética				
Scale Factor	*Min. Neighbours*	*Precisión*	*Recall*	*M*
1,29	17	84,65 %	73,76 %	79,21
1,28	20	87,11 %	70,29 %	78,7
1,3	22	90 %	66,83 %	78,41
1,28	22	88,46 %	68,31 %	78,38
1,29	16	81,62 %	74,75 %	78,18

Tabla 4.1: Mejores 5 configuraciones para maximizar la media aritmética.

Media Armónica				
Scale Factor	*Min. Neighbours*	*Precisión*	*Recall*	*H*
1.29	17	84.65 %	73.76 %	78.83
1.29	16	81.62 %	74.75 %	78.03
1.16	26	78.10 %	77.72 %	77.91
1.28	20	87.11 %	70.29 %	77.808
1.2	22	78.39 %	77.22 %	77.803

Tabla 4.2: Mejores 5 configuraciones para maximizar la media armónica.

Como se puede observar en las tablas 4.1 y 4.2, la configuración SF: 1.29 & MN: 17 (que se apoda desde ahora como la "mejor configuración") fue la ganadora usando ambas métricas, sin embargo, para resolver el problema de detección de rostros en un gran volumen de datos puede que queramos privilegiar un factor por sobre otro, específicamente parece ser que ganar precisión perdiendo un poco de recall es buena idea, debido al hecho de que el volumen de datos es tan grande que perder unos cuantos rostros por ganar confianza entre los rostros detectados ayudará a que la etapa de reconocimiento tenga mejores resultados. Considerando lo anterior, las configuraciones SF: 1.3 & MN: 22 y SF: 1.28 & MN: 20 podrían ser útiles.

Es interesante preguntarse qué efecto tiene variar los valores de SF y MN en la media armónica, en la Fig 4.2 se grafica la media armónica versus el SF y el MN, respectivamente. Del gráfico podemos observar un comportamiento uniformemente distribuido, excepto por la tendencia a aumentar la cota mínima para los valores alrededor de SF: 1.32 y MN: 22, esto no indica que estos valores sean la mejor configuración, sino que esta configuración maximiza el mínimo en cada uno de sus parámetros.

[2]Los resultados de ambas tablas son resultados aproximados.

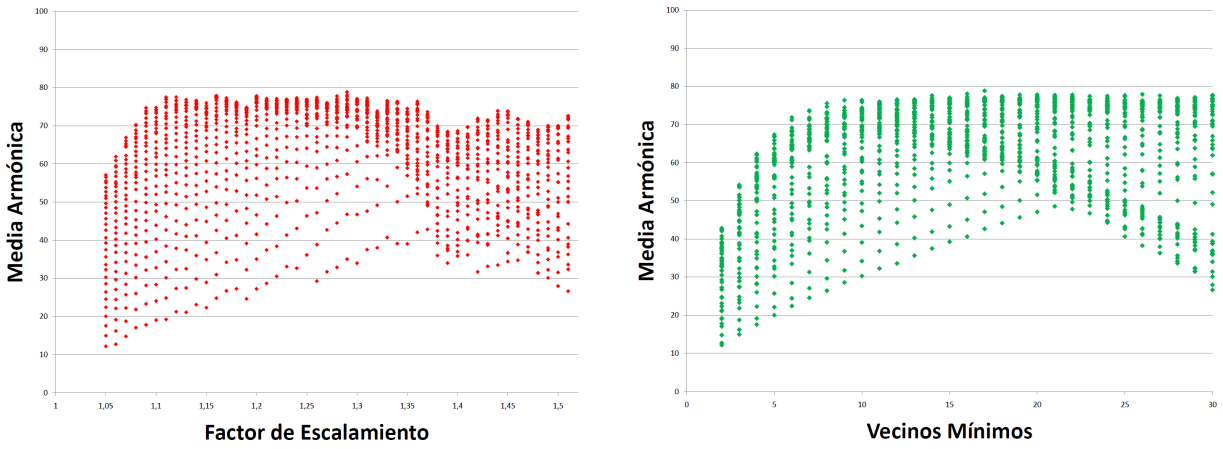

Figura 4.2: Gráfico indicando el impacto de SF y MN en H.

La Fig 4.2 a pesar de ser interesante, no indica el aporte individual de la variación de SF y MN en el recall y la precisión del detector. Para hacer el siguiente experimento, se eligen 3 valores SF mientras que se deja MN como variable libre (y viceversa). Los resultados los encontramos en las Fig 4.3 y 4.4.

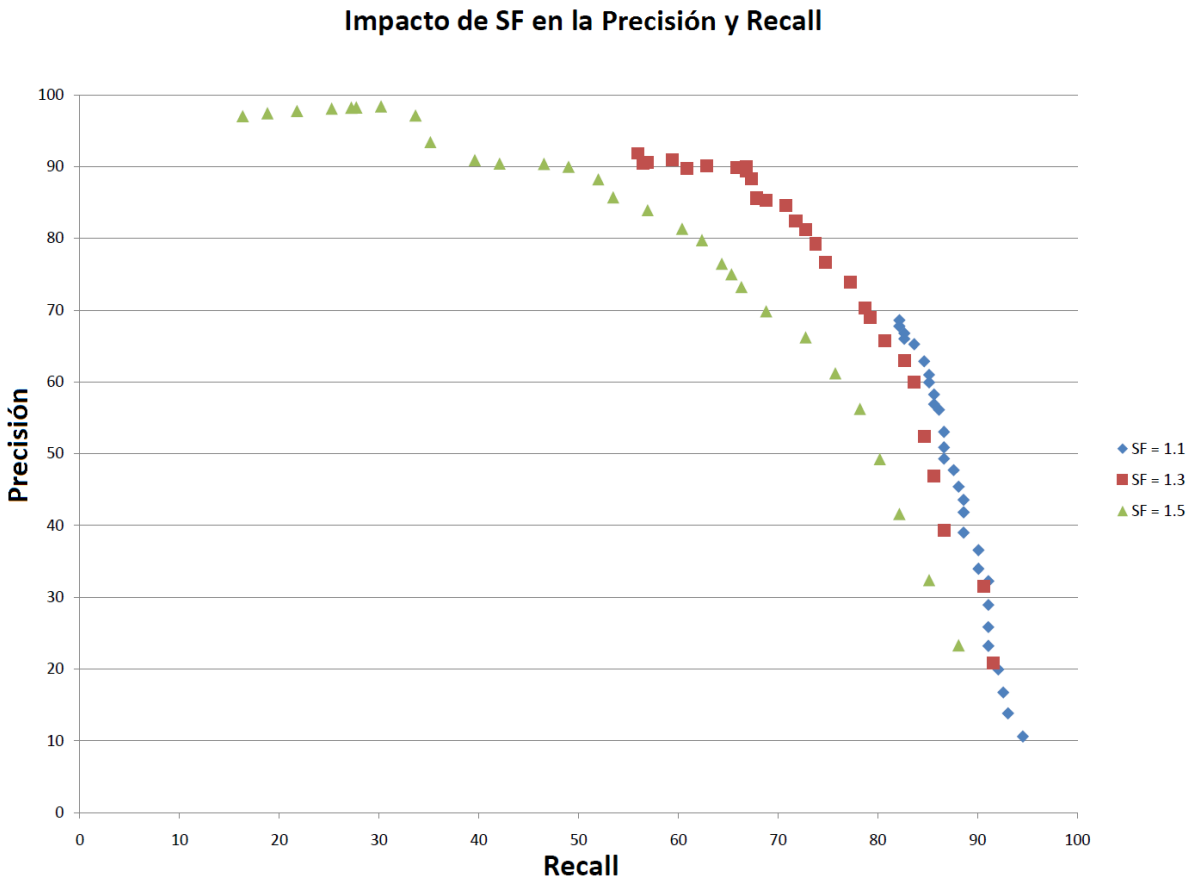

Figura 4.3: Gráfico indicando el impacto de SF en la Precisión y el Recall.

De las Fig 4.3 y 4.4 se pueden sacar importantes conclusiones. Respecto al factor de escalamiento, en la Fig 4.3 se observa que un valor más cercano a 1 tiende a aumentar el

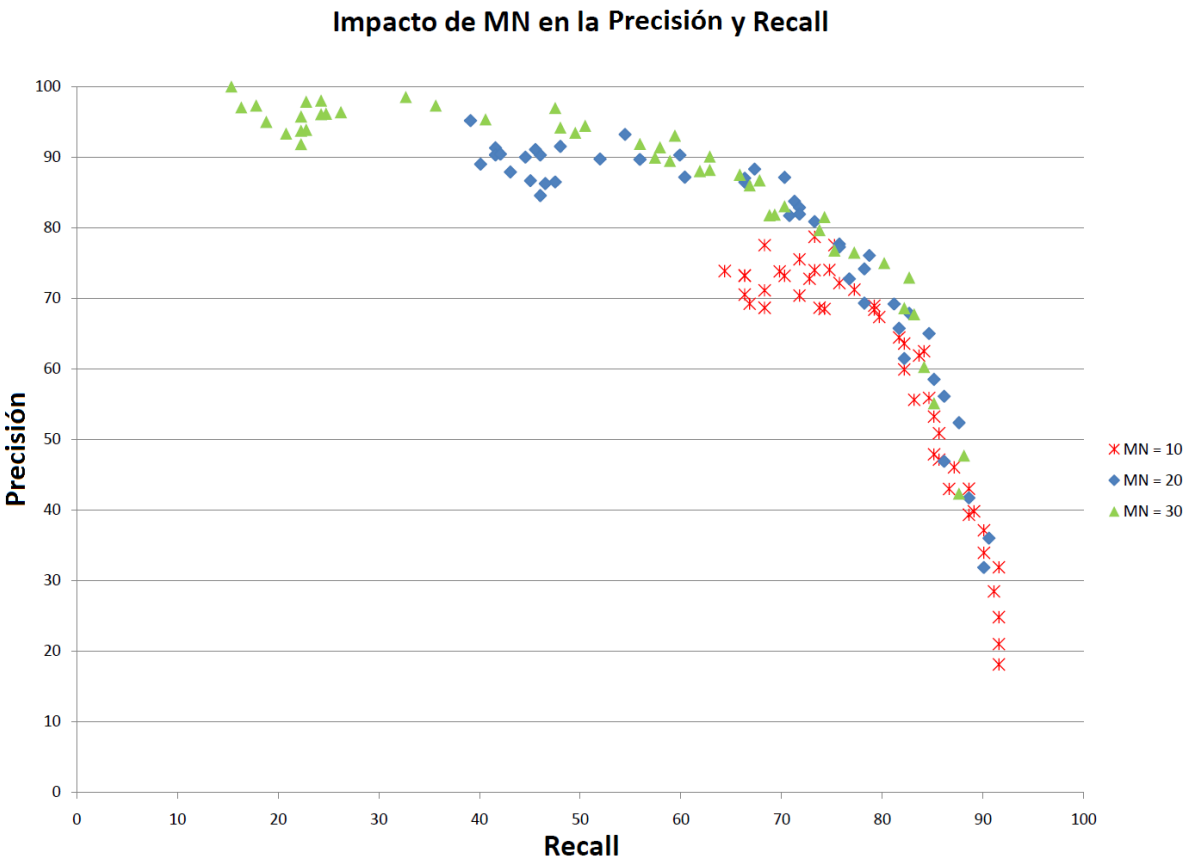

Figura 4.4: Gráfico indicando el impacto de MN en la Precisión y el Recall.

recall pero disminuir la precisión, en cambio un valor más cercano a 1.5 hace tender la curva hacia valores más altos de precisión, sacrificando recall, esta tendencia hace sentido al hecho de que entre más pequeño sea el scale factor, mayor es la cantidad de iteraciones y por lo tanto más candidatos a rostro son considerados. También es interesante observar que de las 3 series los valores con mayor recall no se encuentran tan alejados, en cambio, los con mayor precisión tienen grandes diferencias, este hecho lleva a pensar que el factor de escalamiento afecta más en la precisión del detector en vez del recall.

Similarmente, en la Fig 4.4 se observa que a mayor sea el valor de MN aumenta la precisión sacrificando recall. Estos datos se diferencian de los de la Fig 4.3 en que son más dispersos y no siguen un comportamiento lineal. El comportamiento de MN, similar al de SF, hace sentido al hecho de que entre mayor sea el valor más estricto será el detector, disminuyendo los candidatos de caras que resultan positivos, pero a su vez, aumenta la calidad del detector en términos de precisión.

4.1.3. Análisis de Performance

Paralelo al análisis de precisión y recall es interesante analizar cómo afectan los parámetros a la performance del detector. Para este experimento se mide el tiempo que se demora cada configuración en ejecutar los tests. Se observa que el parámetro de MN **no** afecta la

performance del detector, esto tiene sentido debido a que el detector debe barrer toda la imagen de todos modos, el factor de MN sólo determina un contador de frames contiguos con matches positivos, como se describe en la Sección 1.3.3. En cambio, el SF afecta de manera importante la performance del detector, resultados que se resumen en la Fig 4.5, donde se grafica el tiempo promedio en segundos de ejecutar los tests para cada valor de SF.

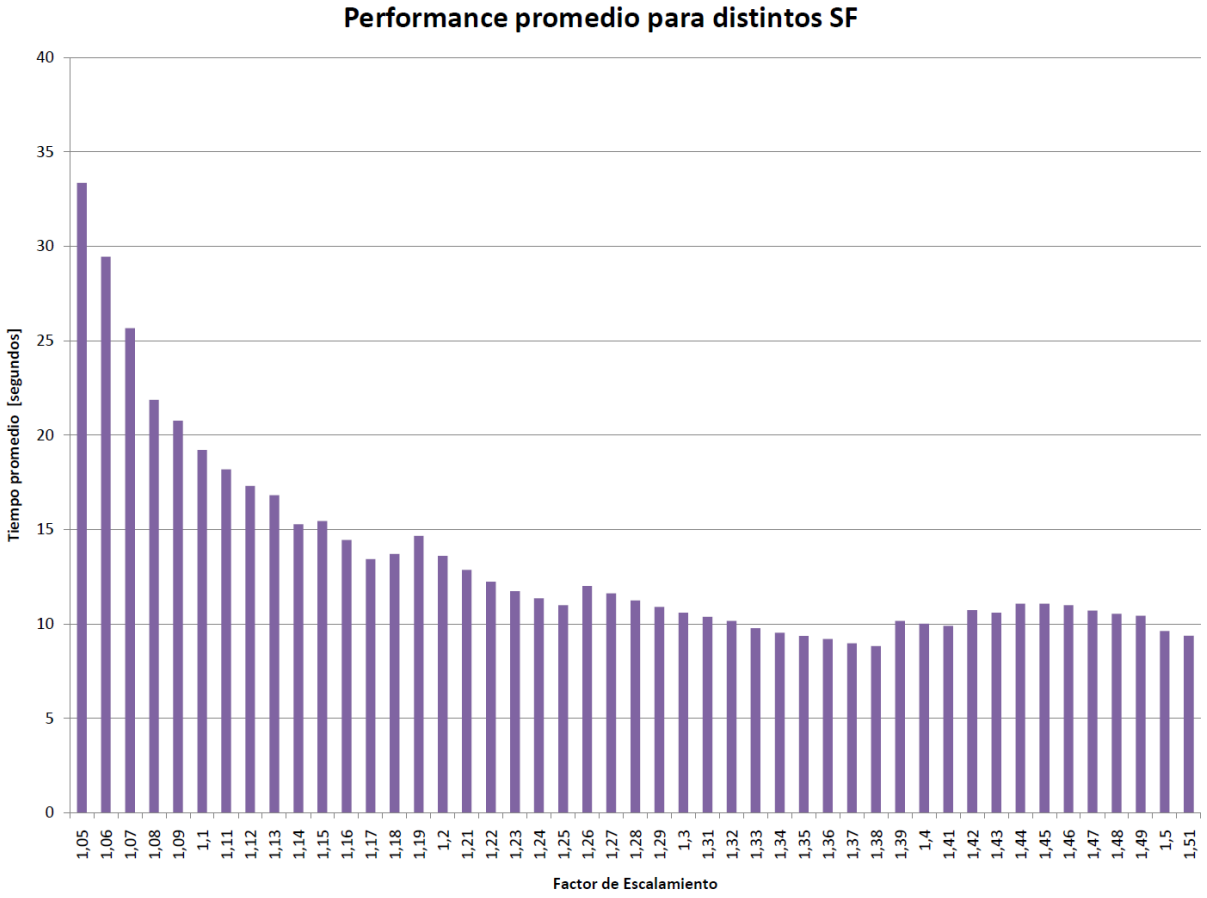

Figura 4.5: Gráfico indicando el impacto de SF en la performance.

Como se observa en la Fig 4.5 entre más cercano sea el valor de SF a 1, más lenta será la ejecución del detector, esto hace sentido ya que a mayor sea el valor de SF mayor será la cantidad de iteraciones sobre la misma imagen. El gráfico muestra que, sin embargo, la curva es descendiente y desciende más rápido al principio que al final, estancándose alrededor de 10 segundos pasado un factor de escalamiento de 1.3.

Como se indica en la sección pasada en las tablas 4.1 y 4.2, la mayoría de las configuraciones que maximizan M y H tienen un SF de alrededor de 1.28, por lo tanto cabe hacerse la pregunta de si es que mayor tiempo de procesamiento mejora el output del detector, esos resultados se entregan en la Fig 4.6, donde se grafica la media armónica de cada configuración del detector versus su tiempo de performance, adicionalmente se indica con una flecha azul el punto que corresponde a la "mejor configuración".

De la Fig 4.6 se puede observar que no hay una correlación directa ni inversa entre la performance con la correctitud, de hecho cabe destacar que la "mejor configuración" está

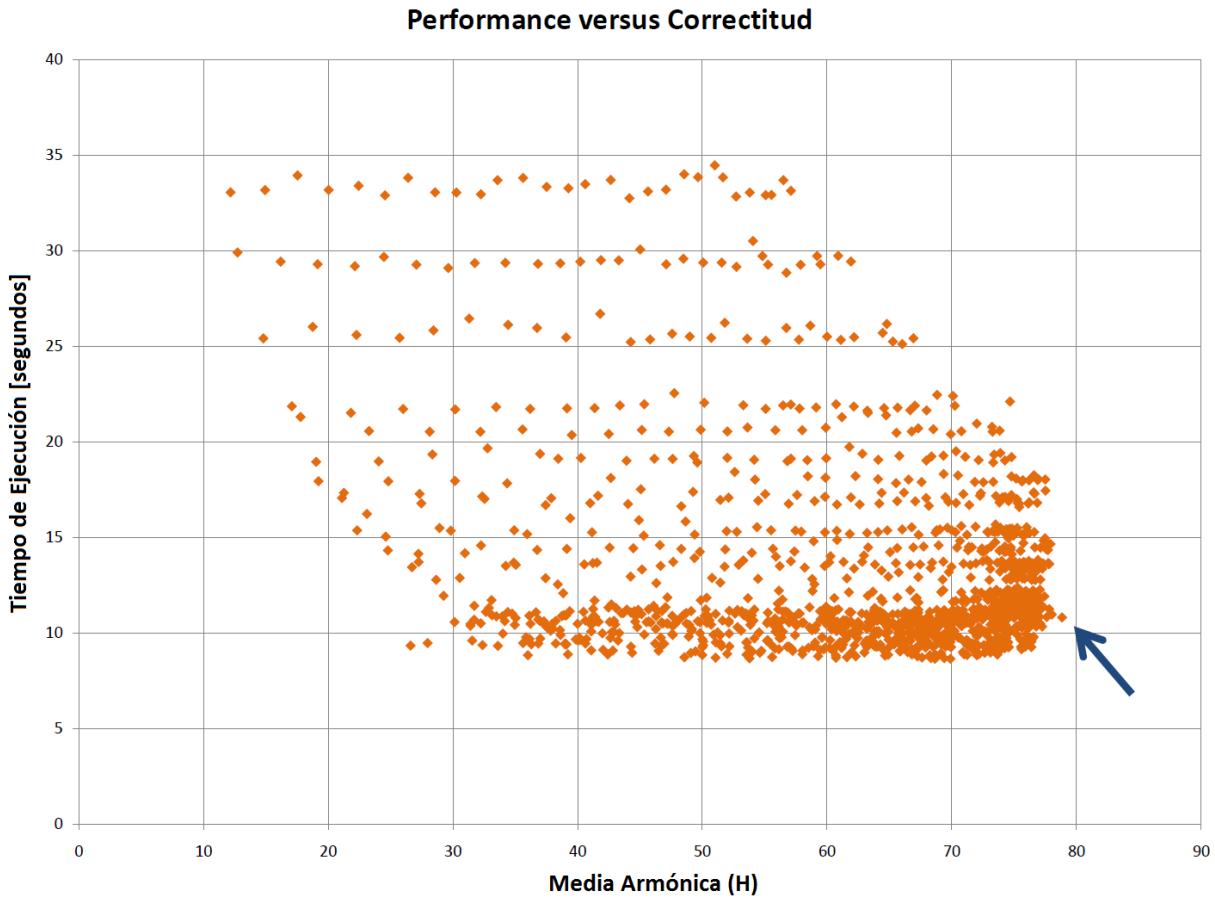

Figura 4.6: Gráfico indicando la correlación entre correctitud y performance.

mucho más cerca del mínimo tiempo de la serie que del máximo, y es la que maximiza la media armónica.

4.2. Evaluación del Proceso de Identificación de Rostros

En esta sección se presentan los resultados del proceso de cálculo de features.

4.2.1. Conjunto de Prueba

Para hacer las siguientes pruebas se utiliza nuevamente un conjunto de evaluación *inhouse* que consiste en imágenes de query de personajes que pertenecen a alguna serie del dataset. El conjunto de prueba consiste en un total de 10 queries (de 10 personajes distintos), las cuales a su vez están compuestas por 5 imágenes de rostros de ese personaje. En la Fig 4.7 se muestra aquellos personajes elegidos para ser queries.

Las imágenes pertenecientes al conjunto de prueba fueron extraídas de Internet usando el

buscador de imágenes de Google.

Los personajes elegidos para ser queries fueron escogidos aleatoriamente, guardando la sutileza de ser personajes con un esquema de colores variado y que su cantidad de apariciones dentro del dataset sea considerable (por lo menos 50).

Figura 4.7: Personajes usados para las queries.

Los personajes utilizados como queries son: (primera fila, de izquierda a derecha) 1.-Kashima Yuu (de *Gekkan Shoujo Nozaki-kun*), 2.-Kinushima Michiru (de *Plastic Memories*), 3.-Lord Cruhteo (de *Aldnoah Zero*), 4.-Orihara Izaya (de *Durarara!!*) y 5.-Megumi Sakura (de *Gakkou Gurashi*), (segunda fila, de izquierda a derecha) 6.-Shion Karanomori (de *Psycho-Pass*), 7.-Shinoa Hiragi (de *Owari no Seraph*), 8.-Makise Kurisu (de *Steins;Gate*), 9.-Asuna Yuuki (de *Sword Art Online*) y 10.-Yuuta Iridasu (de *Punchline*).[3]

4.2.2. Análisis de Precisión

En esta sección se expone el método de evaluación de precisión para el feature y para el mecanismo de búsqueda.

El primer paso en la evaluación es correr el módulo de distancia para cada una de las 10 queries. Cada ejecución resulta en **n** vecinos más cercanos en el dataset, estos **n** vecinos representan los mejores candidatos a ser el personaje que buscamos.

Para determinar la correctitud, primero se determina la precisión promedio (*Average Precision* o AP) en las respuestas positivas para cada una de las queries. Para entender cómo se hace esto, tomemos una query de ejemplo, la que llamaremos *q0* que al igual que las otras queries está compuesta por 5 imágenes de un personaje específico. Al correr el módulo de distancia en q0 este devuelve **n** vecinos más cercanos, los cuales serán denominados x_1 ... x_n,

[3]Las imágenes mostradas en la figura NO son ejemplos de imágenes de query, ya que no son rostros sino más bien bustos de los personajes.

podemos asumir que éstos están ordenados de más a menos votado debido a que son output del módulo de distancia. A continuación lo que se hace es determinar la precisión acumulada en cada uno de los x_i que correspondan a matches positivos, es decir, recorremos cada uno de los x_i y por cada uno de ellos que represente un match positivo, determinamos la precisión **hasta ese momento**. En la Fig 4.8 se diagrama la ejecución de esta técnica. Finalmente se promedian estas precisiones, teniendo una precisión promedio por consulta (AP), las cuales se promedian nuevamente obteniendo un promedio global de precisión llamado Precisión Media Promedio (*Mean Average Precision* o MAP).

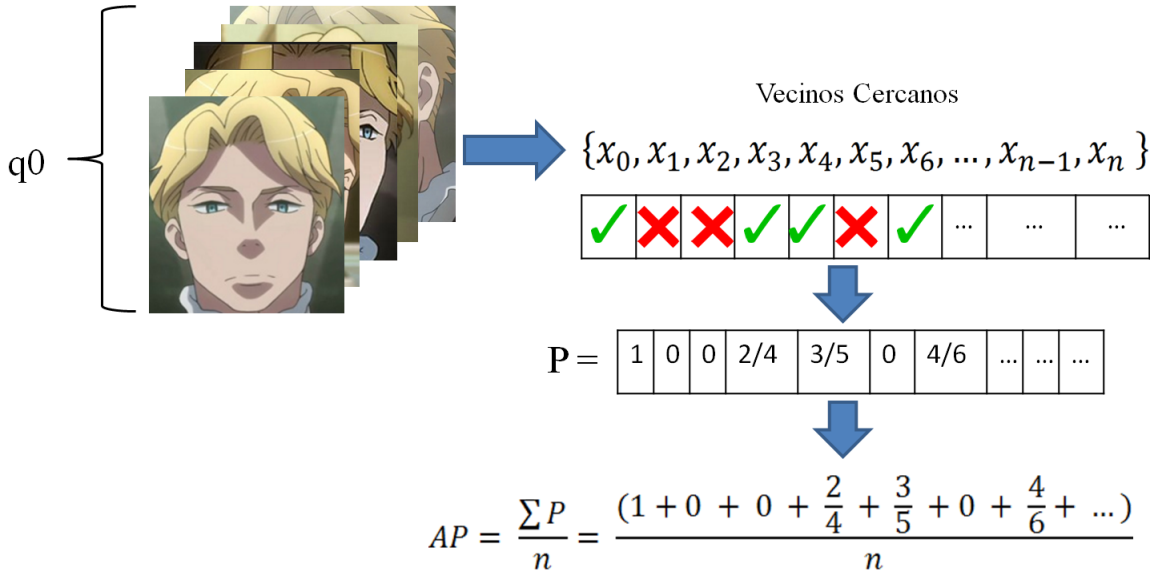

Figura 4.8: Explicación del método de evaluación de features

Sin embargo, si se usa este enfoque, sólo funcionaría si se busca una cantidad de vecinos cercanos grande; esto debido a que la cantidad de apariciones de un personaje principal en una serie supera por mucho las 100, por lo tanto si se eligiera, por ejemplo, 5 vecinos más cercanos, sería fácil obtener una precisión del 100 % cuando puede ser que los siguiente 95 vecinos sean matches negativos, lo que claramente no sería un buen resultado.

Como a cada match positivo se le calcula la precisión acumulada, entre mayor sea n menor aporte van a tener los valores al final. En este proyecto se ha decidido que 50 es suficiente para determinar la calidad de manera fehaciente; además, debido a las limitadas capacidades técnicas y logísticas que cuenta el proyecto, haber hecho una mayor cantidad de vecinos o mayor cantidad de queries hubiese sido inmanejable, pero se reconoce que hubiese mejorado la calidad de la evaluación. Aún así, debido a limitaciones de la librería ocupada para la implementación, se ocupa 32 vecinos más cercanos solamente, se reitera que de haber tenido más tiempo y mejores capacidades técnicas y logísticas se podría haber hecho un mejor análisis. Debido a que sólo disponemos de 32 vecinos y cada personaje en el query tiene (muchas) más apariciones, se ha decidido que se promedie un 0 por cada aparición negativa en las 32 primeras apariciones, comportamiento que se mapea en la Fig 4.8.

Los resultados se exponen en la Fig 4.9 con barras azules, en este caso se ha decidido ocupar 16 bins (en vez de 32) tanto en los features del dataset como en los del input (provocando que HoH-zone sea un descriptor de largo 80). En el gráfico se puede observar que la precisión varía mucho entre clase y clase; en efecto, la consulta por el personaje *7.-Shinoa Hiragi* tiene una precisión muy buena del sobre 70 %, los personajes *9.-Asuna Yuuki*, *5.-Megumi Sakura* y *10.-Yuuta Iridasu* tienen una precisión promedio, y el resto tiene una precisión bastante mala, notablemente *3.-Cruhteo*, *6.-Karanomori* e *4.-Izaya* tienen 0 matches positivos en los 32 primeros vecinos cercanos.

Como completitud y a modo de comparación se ha realizado el mismo proceso pero tomando 32 bins para los features tanto del input como del dataset, los resultados se observan nuevamente en el gráfico de la Fig 4.9 en barras rojas. Los datos para ambos bins se exponen en la tabla 4.3.

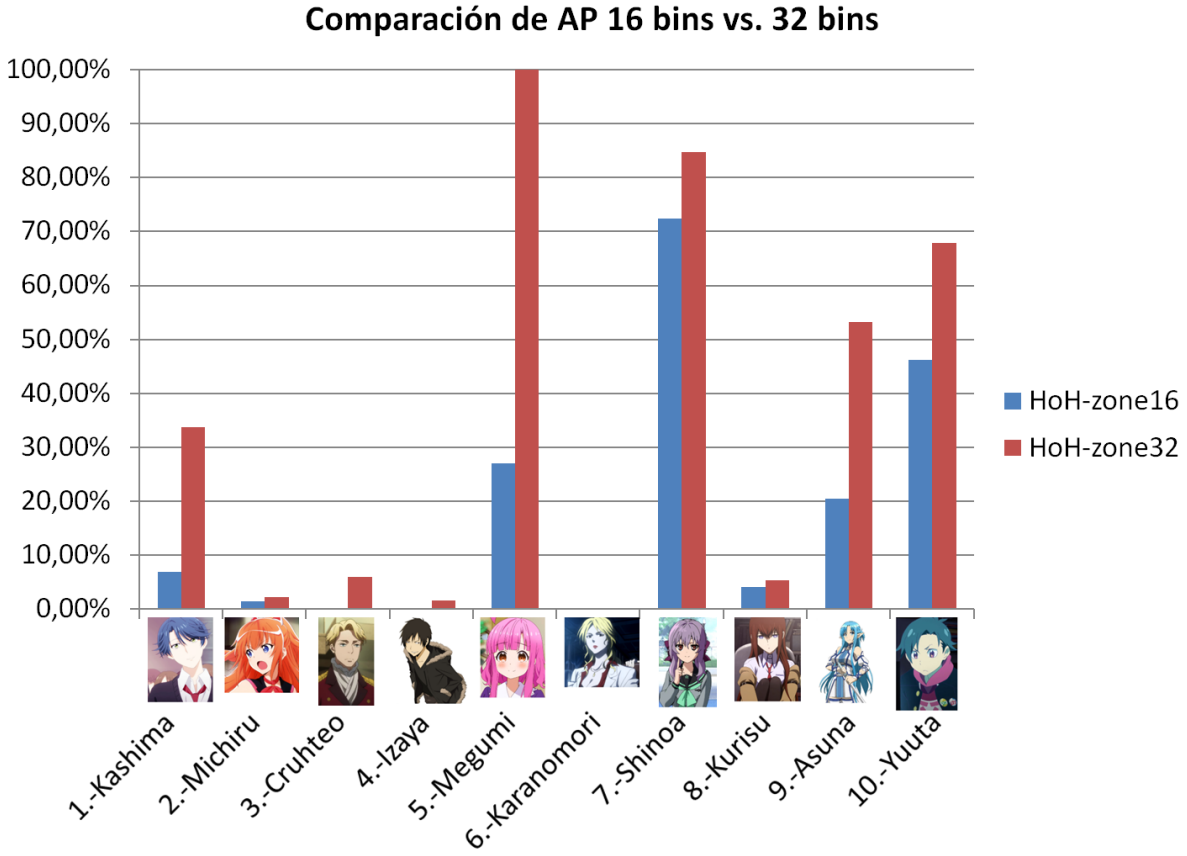

Figura 4.9: Gráfico de AP por clase, 16 y 32 bins.

¿A qué se puede deber el comportamiento mostrado en el gráfico? Fijándose en la Fig 4.7 podemos notar que los 4 mejores resultados son personajes de color de pelo "inusual" o "extravagante", mientras aquellos personajes que sacaron precisión 0 son rubios o de pelo negro. Probablemente, el comportamiento se explica con el hecho de que hay muchos más personajes con color de pelo "regular" que con color de pelo inusual, esto debido a que, como afirmamos anteriormente, el color de pelo inusual es reservado generalmente para los personajes principales, para favorecer que el televidente reconozca al personaje rápidamente en distintas situaciones, mientras que el resto de personajes secundarios y de fondo tienen

una coloración de pelo "normal", esto provoca que sea más fácil confundir dos personajes con coloración regular de pelo.

Para *3.-Lord Cruhteo* y *6.-Karanomori Shion*, los matches entregados por el detector fueron principalmente personajes con el pelo rubio; matches, que como se afirmó anteriormente, sólo fueron matches negativos, lo que se explica con el argumento del párrafo anterior. Sin embargo, a diferencia de los personajes rubios, para el personaje *4.-Orihara Izaya*, los matches no corresponden a personajes de sus características, hay matches de personas rubias, morenas y pelirrojas, y en algunos casos ni siquiera corresponden a personajes sino a objetos. Posiblemente, la razón detrás esto es que el color negro en el espacio HSV corresponde a un punto con S (saturación) y V (valor) igual a 0 y H (Tono o Matiz) cualquiera, debido a que HoH-zone se basa en histogramas de hues, el negro calza con todos los hues y por lo tanto calza con muchos más rostros que otras coloraciones de pelo.

Comparación de AP		
	HoH-zone16	*HoH-zone32*
1.-Kashima	6,85 %	33,74 %
2.-Michiru	1,5 %	2,24 %
3.-Cruhteo	0 %	5,9 %
4.-Izaya	0 %	1,56 %
5.-Megumi	27,06 %	100 %
6.-Karanomori	0 %	0 %
7.-Shinoa	72,31 %	84,72 %
8.-Kurisu	4,17 %	5,4 %
9.-Asuna	20,54 %	53,13 %
10.-Yuuta	46,11 %	67,89 %

Tabla 4.3: Comparación de AP, HoH-zone de 16 y 32 bins.

En el gráfico y en la tabla podemos observar que con 32 bins todas las búsquedas mejoraron su precisión, exceptuando la de *6.-Shion Karanomori* que quedó en 0 de todas maneras. También se observa que las tendencias que veíamos con 16 bins se mantienen con 32 bins también. La desventaja de usar 32 bits es que ahora el descriptor tiene largo 160 en vez de 80, provocando que se ocupe el doble de espacio en disco para guardar los descriptores.

Debido a lo trabajosa que resulta la evaluación de la búsqueda, no se probó con otra iteración del feature, probar una mayor cantidad de bins queda considerado para posibles trabajos futuros.

Finalmente, queda responder la pregunta de cuál de las dos iteraciones es la mejor. Para resolver esta duda y como se indicó anteriormente se calcula MAP para ambas series de datos, adicionalmente, a cada gráfico se le agrega su porcentaje de error, resultados que se resumen en el gráfico de la Fig 4.10 y en la tabla 4.4.

Alcanzando aproximadamente un MAP de 35 % versus uno del 18 %, la esperanza favorece al feature de 32 bins, sin embargo, como se observa en el gráfico de la Fig 4.10, considerando los porcentajes de error, los rangos se traslapan; a pesar de eso, gracias al gráfico de la Fig 4.9 podemos verificar que el descriptor de 64 bins mejora o deja igual la precisión de todas las

consultas, en efecto, ninguna disminuye su precisión; es debido a este motivo que se afirma que el feature de 64 bins es mejor que el de 32 para la consultas indicadas.

Nuevamente, debido a limitaciones logísticas no se pudo hacer para otra iteración del feature, pero se especula que una iteración del feature con 64 bins o más puede resultar en mejores MAPs, teniendo en consideración que el tamaño del descriptor crecería al doble, requiriendo más espacio para su almacenamiento.

Comparación de MAP		
	HoH-zone16	*HoH-zone32*
Media	17,85 %	35,46 %
Desviación Estándar	23,21 %	36,39 %
Error	14,38 %	22,55 %
Mínimo	3,22 %	12,9 %
Máximo	32,24 %	58,01 %

Tabla 4.4: Datos finales, HoH-zone de 16 y 32 bins.

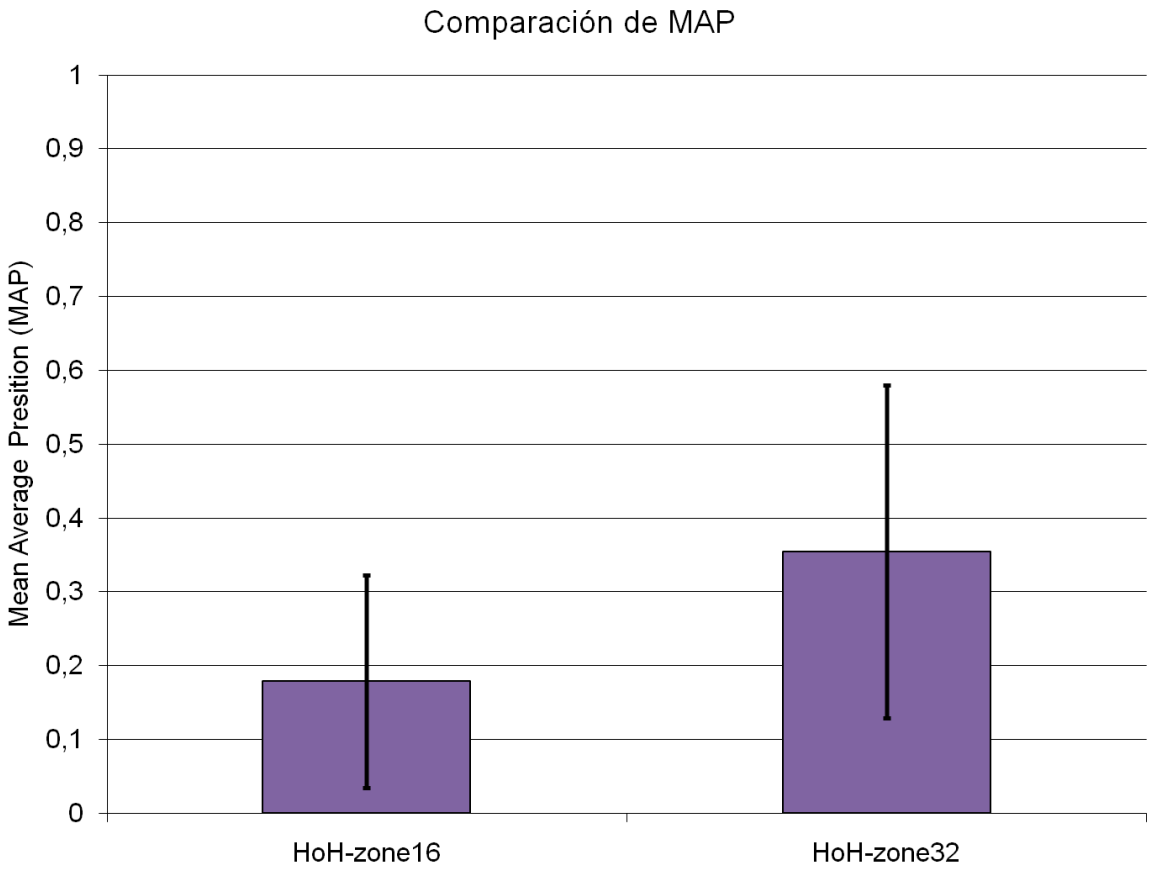

Figura 4.10: Gráfico de comparación de MAPs.

Conclusión

En este proyecto se enfrenta el problema de *Cartoon Character Detection and Recognition*; problema que, como ya se ha afirmado anteriormente, responde a una necesidad que es la inminente creación de herramientas basadas en visión por computador que logren trabajar con material animado, que a su vez se corresponde con una tendencia creciente al uso de aplicaciones basadas en técnicas de visión por computador y aprendizaje computacional.

Como se discute en el documento, se ha propuesto un proceso de solución de 4 pasos; la búsqueda de keyframes, una fase de detección basada en imágenes, un cálculo de features visuales y por último una fase de búsqueda y votación.

También se discute que los verdaderos objetivos del proyecto no sólo engloban la creación de la aplicación capaz de cumplir estos requisitos, sino la obtención de resultados competitivos en cada una de las subfases del proyecto, el cual es un objetivo bastante más desafiante.

Se expuso también su implementación hecha en Python, cómo se le dedicó un módulo a cada uno de los subprocesos que componen el proyecto y cómo éstos fueron implementados para resolver la problemática.

Refiriéndose al cumplimiento de los objetivos del proyecto, podemos decir que la aplicación construida resuelve el problema principal, la aplicación cuenta con todas las funcionalidades necesarias, además de proveer una interfaz relativamente fácil de usar, documentación y tests cuando fuesen necesarios.

Ahora, refiriéndose a la calidad de esta solución; si se observan los resultados del detector, estos superan con creces los resultados que se tenían presupuestados, con una configuración logrando aproximadamente 85 % de precisión y 74 % de recall superando satisfactoriamente el 61 % de precisión alcanzado por Chau en [3].

Dentro de los experimentos que se realizaron, se analizó el impacto de variar los parámetros del detector en la calidad del output, resultado que es muy valioso para futuros intentos de solución del problema. Por último se verifica que la mejor configuración, contrario a lo que se pensaría, es una de las más eficientes en términos de performance.

Habiendo dicho esto, hay varias mejoras que pueden ser desarrollados a futuro para mejorar el proceso de detección; definitivamente, mayor cantidad de ejemplos de entrenamiento producirían un mejor detector, mejora que en este proyecto no pudo ser implementada debido a la falta de tiempo y capacidades logísticas.

En definitiva, se concluye que el detector funciona extremadamente bien considerando el limitado número de ejemplos con que fue entrenado; a modo de comparación recordar que Imager::AnimeFace entrenó su detector con más de 70.000 ejemplos positivos y 300.000.000 de ejemplos negativos versus los 850 positivos y 500 negativos de este proyecto, y se logró alcanzar mejores precisiones (tomando como referencia el cálculo hecho por Chau en [3]). Asimismo se concluye que usar un enfoque de detección basado en imágenes puede generar buenos resultados para el contexto en el que se da el proyecto.

Por otra parte, considerando los resultados de feature y votación, verificamos que funcionan muy bien para las búsquedas de personajes de coloración extravagante (que abundan en el animé), pero no funcionan tan bien para personajes de coloración normal, esto se debe a la manera en que se comportan los features basados en tonos de color, concepto que es profundizado en la sección correspondiente. A modo de comparación, nuevamente, podemos comparar nuestros resultados a los de Chau, el cual alcanza una precisión promedio de 52,8 %.

Sin embargo, una vez visto los resultados de los features, las posibles mejoras no son claras a simple vista; como se discute es la Sección 1, el pre-filtro por color de piel no resulta necesario habiendo tenido un output tan bueno del detector y haber pesado más los pixeles correspondientes a pelo hubiese mejorado la precisión global, pero no hubiese mejorado la precisión de aquellas clases difíciles de reconocer, por otro lado, se propone combinar HoH-zone con un feature de otra familia distinta, como por ejemplo un feature de bordes.

También, a futuro se pueden probar otras técnicas de votación, o funciones de distancia distintas, las cuales en definitiva pueden hacer variar el resultado final del proyecto.

En definitiva, los resultados indican que para cierto tipo de búsquedas, los descriptores visuales basados en color representan una buena solución, y los resultados expuestos avalan esta afirmación.

Este proyecto genera un impacto en el análisis de media animada, principalmente debido a que se verifica la factibilidad de solución utilizando técnicas como las utilizadas en este proyecto. También se desarrolla una métrica de evaluación de estas técnicas, lo que puede resultar útil como medida comparativa a futuro.

En otra instancia del proyecto, se puede intentar reproducir el mismo proceso de solución, ahora aplicado a la animación en general y no acotado solamente a animación japonesa.

Este proyecto no sólo enseña el funcionamiento de mecanismos basados en imágenes para la resolución del problemas de reconocimiento animado, sino que también enseña métodos de evaluación de estas técnicas. Ser riguroso en la evaluación es importante para poder hacer comparaciones a futuro con otras técnicas.

En el futuro, se puede explorar la factibilidad de resolver este problema pero usando otro tipo de técnicas, ya no basadas en el proceso clásico de cálculo de features sino más bien en redes neuronales, éstas son conocidas también como *Deep Learning*, las cuales han demostrado ser la opción ganadora para casi todo tipo de medios en el último tiempo, sin embargo, hasta el momento nadie se ha aventurado a ver cómo se comportan estas redes neuronales en el reconocimiento de personajes animados.

Bibliografía

[1] Masayuki Arai. Feature extraction methods for cartoon character recognition, 2012.

[2] John S Boreczky and Lawrence A Rowe. Comparison of video shot boundary detection techniques. *Journal of Electronic Imaging*, 5(2):122–128, 1996.

[3] Herman Chau. Face detection and recognition of drawn characters, 2014.

[4] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[5] Y. Takai H. Kawatani, H. Kashiwazaki and N. Takai. Anime degree evaluation by feature extraction of animation characters, 2008.

[6] Igata. `http://www.kodamalab.hc.uec.ac.jp/works/pdf/2006/MangaAnalysis.pdf`.

[7] Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang, and Stan Z Li. Learning multi-scale block local binary patterns for face recognition. In *Advances in Biometrics*, pages 828–837. Springer, 2007.

[8] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[9] Timo Ojala, Matti Pietikainen, and David Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, volume 1, pages 582–585. IEEE, 1994.

[10] Constantine P Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. In *Computer vision, 1998. sixth international conference on*, pages 555–562. IEEE, 1998.

[11] Kazi Tanvir Ahmed Siddiqui and Abu Wasif. Skin detection of animation characters.

[12] Kohei Takayama, Henry Johan, and Tomoyuki Nishita. Face detection and face recognition of cartoon characters using feature extraction. In *Image, Electronics and Visual*

Computing Workshop, page 48, 2012.

[13] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[14] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[15] HongJiang Zhang, Atreyi Kankanhalli, and Stephen W Smoliar. Automatic partitioning of full-motion video. *Multimedia systems*, 1(1):10–28, 1993.