Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Tesis de Licenciatura en Ciencias de la Computación

Detección y Reconocimiento de Caras

Alumno	L.U.	Correo electrónico
Morelli Andrés, Ariel	90/01	amorelli@dc.uba.ar
Padovani, Sebastián	183/99	spadovan@dc.uba.ar

Directores:

MSc. Mariano Tepper
Dra. Marta Mejail

Buenos Aires, Marzo de 2011

Resumen

La detección y el reconocimiento de rostros son temas que están siendo ampliamente estudiados debido a la gran cantidad de aplicaciones que tienen. Por ejemplo, muchas cámaras fotográficas digitales, antes de tomar una fotografía, detectan los rostros presentes en la escena para mejorar el enfoque y la nitidez. También se utiliza la detección de rostros en aplicaciones de seguridad y seguimiento de personas, entre otras. Además, podemos encontrar sistemas de reconocimiento de rostros en páginas web de redes sociales, software de administración de fotografías, sistemas de control de acceso, por nombrar algunos de ellos.

El reconocimiento de rostros tiene la ventaja que, a diferencia de otros sistemas biométricos, es un método no invasivo. Es decir, que no se requiere ningún contacto físico para tomar una muestra. Es más, con la tecnología disponible, las imágenes pueden obtenerse con suficiente resolución y sin que el individuo se percate de ello (como por ejemplo, mediante las cámaras de seguridad instaladas en los aeropuertos, locales e incluso en la vía pública).

El objetivo de esta tesis es construir un sistema capaz de detectar las caras presentes en una imagen o secuencia de video y luego poder determinar la identidad de dichas personas. Para ello, presentamos el algoritmo de detección de rostros propuesto por Viola y Jones [46] junto con todas las mejoras que surgieron a partir del mismo. Luego, abordamos el reconocimiento de rostros utilizando un novedoso método que tiene su fundamento en recientes avances en la teoría de *compressed sensing*, el cual modificamos para que sea capaz de detectar y descartar las zonas ocluidas del rostro, mejorando el reconocimiento ante la presencia de oclusión.

Abstract

Face detection and recognition are widely studied nowadays due to the large number of applications they have. For instance, many digital cameras before taking the picture detect the faces present in the scene in order to improve focusing and sharpness. Face detection is also used in security applications and people tracking systems, among others. In addition, face recognition systems are found in social networking sites, photo management software, access control systems, to name some.

Face recognition has the advantage that, unlike other biometric systems, is a noninvasive method. In other words, it requires no physical contact to take a sample. Moreover, with the technology available, images can be obtained with sufficient resolution and without the individual realizing it (for example, through security cameras installed at airports, shops or even public spaces).

The aim of this thesis is to build a system to detect the faces present in an image or video sequence, and then to determine the identity of such individuals. To this end, we introduce the face detection algorithm proposed by Viola and Jones [46] along with all the improvements that arose from it. Then, we tackle face recognition using a novel method based on the latest technological advances in the theory of *compressed sensing*, which we modified to make it able to detect and discard occluded areas, improving recognition on the presence of occlusion.

Índice

1. Introducción **9**

2. Detección de Rostros **15**
2.1. Algoritmos de Entrenamiento y Detección 18
2.1.1. Imagen Integral 19
2.1.2. Algoritmo de Boosting 21
2.1.3. Cascada de Clasificadores Fuertes 22
2.1.4. Algoritmo de Detección 26
2.2. Mejoras sobre los Algoritmos de Entrenamiento y Detección . 29
2.2.1. Conjunto de Features Extendido 29
2.2.2. Post-Optimización de Clasificadores Fuertes 31
2.2.3. Gentle Adaboost 31
2.2.4. Stump vs CART 34
2.2.5. Árbol de Clasificadores Fuertes 36
2.2.6. Escalado de Features en la Detección 40
2.3. Consideraciones Generales 40

3. Reconocimiento **43**
3.1. Reconocimiento como una descomposición lineal esparsa . . . 44
3.1.1. Modelando el Ruido Gaussiano 48
3.1.2. Selección de *features* 49
3.1.2.1. Random Faces 51
3.2. Clasificación 51
3.3. Reconocimiento en Presencia de Oclusión 53
3.3.1. Particionar el problema 55
3.3.2. Detección de la Oclusión 56
3.4. Síntesis general 64

4. Resultados **67**
4.1. Bases de datos de caras 67
4.2. Detección 68
4.3. Reconocimiento 76
4.3.1. Sin oclusión 76

4.3.1.1. Resultados usando *Extended Yale B* 76
4.3.1.2. Resultados usando *AR Database* 79
4.3.2. Con oclusión 81
4.3.2.1. Modelo Simple 83
4.3.2.2. Modelo Particionado 85
4.3.2.3. Detección de la oclusión 88
4.4. Pruebas de integradoras 92

5. Conclusiones **97**

Bibliografía **99**

Capítulo 1

Introducción

La presente tesis de licenciatura trata del reconocimiento de personas a partir de una imagen o video. En la actualidad existen diversos métodos automáticos para el reconocimiento de personas. Al estudio de estos métodos se lo llama biometría, palabra que deriva de los términos griegos "bios" de vida y "metrón" de medida. La biometría se basa en el estudio de rasgos físicos y conductuales que permiten identificar al individuo y analizar su comportamiento. Para ello, se utilizan diversas técnicas matemáticas y estadísticas. Algunos de los rasgos físicos que se han estudiado hasta el momento son el iris, la retina, el rostro, las manos y las huellas dactilares entre otras.

El reconocimiento de personas tiene como motivación el estudio de la actividad humana y diversas aplicaciones comerciales. En este trabajo se estudia el reconocimiento de las personas a través de su rostro. Cada vez son más las aplicaciones de software y dispositivos electrónicos de uso corriente que incorporan esta tecnología, entre los que podemos mencionar: vigilancia automática de grupos de personas, seguridad en aeropuertos, páginas web de redes sociales, software embebido en cámaras fotográficas, software de procesamiento de imágenes y administración de fotografías, análisis de videos, diseño de interfaces de computadoras (Human Computer Interface – HCI), sistemas de seguridad y confort en vehículos y hogares.

La imagen del rostro humano puede tener grandes variaciones intra-sujeto (cambios en el mismo individuo) que hacen difícil el desarrollo de un sistema de reconocimiento. Las variaciones pueden ser, por ejemplo, la posición de la cabeza en el momento de tomar la imagen, la iluminación, la expresión del rostro (risa, enojo, etc.), ocultamiento de ciertas partes del rostro debido al uso de accesorios como lentes de sol y bufanda, bellos en la cara (bigotes, barba, pelo largo, etc.), y cambios en los rasgos dados por el paso del tiempo. Por otro lado, las variaciones inter-sujeto (diferencias entre distintos individuos) pueden ser muy pequeñas entre dos personas de rasgos similares.

Dada una imagen o cuadro de video, un sistema reconocedor de rostros debe primero detectar los rostros presentes en la imagen y luego reconocerlos. Con el término *detección* nos referimos a localizar en qué regiones de la imagen hay una cara, mientras que con el término *reconocimiento* nos referimos al proceso que, dada una cara, identifica al individuo correspondiente.

La detección de rostros es parte del problema de clasificación de objetos. Existen dos enfoques distintos en las técnicas de aprendizaje de clases de objetos: aprendizaje discriminatorio (*Discriminative Learning*) y aprendizaje generativo (*Generative Learning*) [45]. En el primero, se busca responder a la siguiente pregunta: "dada una imagen a evaluar, ¿contiene el tipo de objeto buscado o no?". Básicamente, el aprendizaje discriminatorio trata de aprender un modelo que sea capaz de distinguir correctamente entre muestras positivas (en nuestro caso, caras) y muestras negativas (en nuestro caso, todo lo que no sea cara). Por otro lado, en el aprendizaje generativo, se busca responder la pregunta: "dado un modelo del objeto, ¿cuán factible es que la imagen a evaluar pueda generarse a partir del modelo?". El aprendizaje generativo busca aprender un modelo que sea capaz de generar todos los objetos de una clase, en contraste con el aprendizaje discriminatorio que busca aprender los umbrales y diferencias entre dos o más clases de objetos. El modelo generativo es computacionalmente más complejo y, en la práctica, el modelo discriminatorio tiene un mejor desempeño.

Una de las técnicas discriminatorias más conocidas es el *boosting* [19] [39]. El *boosting* combina clasificadores muy sencillos, denominados *clasificadores débiles*, para formar un "comité de votación". Un clasificador débil tiene un desempeño levemente mejor que el azar, pero combinando varios clasificadores débiles a través de *boosting* se obtiene un desempeño en conjunto sorprendente. El *boosting* va entrenando clasificadores débiles en forma consecutiva y los combina en lo que se llama un clasificador fuerte. *Adaboost* (de *Adaptative Boosting*) [21] es una variante de *boosting* en la que cada clasificador débil es "adaptado" para darle mayor importancia a los objetos mal clasificados por el clasificador débil anterior. En [46], Viola y Jones usan *Adaboost* para construir un detector de caras extremadamente eficiente, permitiendo realizar la detección en tiempo real.

Un sistema de reconocimiento de rostros puede implementarse en distintos escenarios que requieren de distintos tipos de solución. Uno de ellos es el de *Verificación de Rostros*. Este tipo de solución requiere verificar que la persona es quien dice ser. Para realizar esta verificación es necesario comparar la imagen de la cara del sujeto con una representación de la cara, almacenada en el sistema, de la persona que dice ser. Un escenario distinto es el de *Identificación de Rostros*, en donde se debe determinar cuál de las

personas representadas en una base de datos de caras es la persona a identificar. Para ello se requieren muchas comparaciones entre la cara consultada y las almacenadas. Otro posible escenario es el de una *Lista de Observación* (*Watch List*), en donde el universo de personas es abierto, es decir, el individuo puede o no estar representado en la base de datos. Se compara el rostro de la persona con aquellos almacenados en la base de datos y se le asigna a cada comparación un puntaje de similitud. Si el puntaje de similitud de una persona almacenada supera un umbral dado, se considera que la persona a consultar se encuentra representada en la base de datos y su identidad es la de la persona correspondiente a dicho puntaje. Este último escenario es el que vamos a tratar en este trabajo.

Actualmente, las técnicas de reconocimiento de caras basadas en imágenes pueden ser categorizadas en dos grupos principales de acuerdo a la representación de la cara que utilizan: *Basadas en Apariencia*, que realizan el reconocimiento en base a las características importantes y texturas de la cara, y *Basadas en Modelos*, que utilizan la forma y textura de la cara en conjunto con un modelo de dos o tres dimensiones.

En los métodos basados en apariencia, la imagen es vista como un vector de gran dimensión y, por lo tanto, como un punto en un espacio vectorial de gran dimensión. Muchos de estos métodos, usan técnicas estadísticas para analizar la distribución de los vectores correspondientes a imágenes de caras en el espacio vectorial de imágenes completo. El espacio formado únicamente por vectores de caras se denomina espacio de caras. Luego, obtienen una representación efectiva y eficiente dentro de un espacio reducido denominado espacio de características (*feature space*). Dada una imagen de una cara a evaluar, la similitud entre las muestras almacenadas y esta última, se determina en el espacio de características. Entre los métodos basados en apariencia, están aquellos que usan un mapeo lineal entre el espacio de caras y el espacio de características, como por ejemplo, PCA (*Principal Component Analysis*)[42], también conocido como *Eigenfaces* [44], ICA (*Independent Component Analisis*)[25] y LDA (*Linear Discriminant Analysis*)[17], y también aquellos que usan un mapeo no-lineal, como KPCA (*Kernel Principal Component Analysis*[40] y LLE (*Locally Linear Embedding*)[37]. Una de las ventajas de los métodos basados en apariencia es que se transforma el problema de reconocimiento de rostros en un problema a resolver en un espacio en el que se conocen muchas herramientas estadísticas. Otra de las ventajas es que estos métodos se pueden aplicar tanto en imágenes de alta resolución y buena calidad como en imágenes de baja resolución y mala calidad.

Los métodos basados en modelos pretenden construir un modelo de la cara humana que sea capaz de capturar las variaciones que puedan presentarse

en un rostro. Por lo general constan de tres pasos. En primer lugar, se construye el modelo. Luego, se ajusta el modelo al rostro de la imagen a evaluar. Por último, se compone un vector de características usando los parámetros empleados para ajustar el modelo (*model fitting*), que luego se utiliza para calcular la similitud entre el rostro consultado y las muestras de caras almacenadas. La construcción del modelo se basa en los conocimientos previos que se tienen del rostro humano, por ejemplo, la posiciones y distancias relativas entre los distintos componentes de la cara. Algunos ejemplos de este tipo de algoritmos son: *Elastic Bunch Graph Matching* [47], AAP (*Active Appearance Model*) [14] [16] y *3D Morphable Model* [7] [6] [5]. Una de las ventajas de los métodos basados en modelos es que el modelo posee una estrecha relación con una cara real y se pueden aplicar los conocimientos del rostro humano. Otra ventaja es que se pueden modelar mejor las variaciones que se pueden dar en el rostro, como cambios de pose, iluminación y expresión.

El método que se propone estudiar en esta tesis tiene sus fundamentos en algunos descubrimientos recientes realizados en el campo de la teoría de muestreo y reconstrucción de señales [11] [12]. Se puede considerar al método dentro de la categoría de aquellos basados en apariencia, pero difiere considerablemente del resto, ya que en vez de comparar la similitud entre la cara a evaluar y las caras de la base de datos, usa estas últimas para reconstruir el rostro a evaluar, como si se tratara de reconstruir una señal.

Según el estudio de señales, una señal dada puede ser representada mediante una combinación lineal de muestras de señal. A dichos elementos base o muestras de señal los denominamos átomos. Decimos que una combinación lineal de una colección de átomos provenientes de diferentes señales es esparsa, cuando gran parte de los coeficientes de la combinación lineal son ceros.

Uno de los descubrimientos realizados en el estudio de señales establece que, cuando la representación óptima de una señal es suficientemente esparsa, el problema de hallar dicha representación puede ser modelado como un problema de optimización con restricciones convexas y resolverse de manera eficiente [15].

En [48], se aplica este resultado al reconocimiento de caras y se presenta un método de reconocimiento novedoso y robusto incluso ante la presencia de cierto porcentaje de ruido gaussiano y oclusión localizada. En la presente tesis estudiamos y profundizamos este método de reconocimiento, introduciendo cambios o agregados que conllevan mejoras en el reconocimiento ante la presencia de oclusión localizada. Cuando cierta región de la cara se encuentra ocluída, la misma introduce "ruido" en el algoritmo de reconocimiento, haciendo que el mismo pueda fallar o le sea más difícil encontrar

una solución. En este trabajo se propone un algoritmo de reconocimiento que consta de dos etapas, la primera, que denominamos *detección de la oclusión*, divide las porciones ocluidas de la imagen de las no ocluidas de tal manera de introducir menos error en el reconocimiento. Luego, en la segunda etapa, usando solo las porciones no ocluidas de la imagen, se obtiene la identidad del sujeto. Los resultados que se obtienen al emplear la detección de oclusión propuesta son claramente mejores.

En el capítulo 2 presentamos el método de detección de caras propuesto por Viola y Jones, el cual está basado en un algoritmo de aprendizaje que requiere de una etapa de entrenamiento previa para poder realizar la detección. Además analizamos las distintas mejoras que se fueron realizando sobre el algoritmo, y que se incluyen en el detector que utilizamos.

En el capítulo 3 estudiamos el método de reconocimiento de caras presentado en [48] y proponemos un método que permite obtener una mejor tasa de reconocimiento cuando los rostros se encuentran parcialmente ocluidos, ya sea por anteojos oscuros, barba o bufanda entre otros.

En el capítulo 4 presentamos las pruebas ejecutadas junto con sus resultados y el análisis de los mismos. En primer lugar se presentan las bases de datos de caras utilizadas. Luego se describen los resultados obtenidos con el algoritmo de detección, seguidos de los obtenidos con el algoritmo de reconocimiento. En último lugar se presentan las pruebas realizadas integrando ambos sistemas (de detección y reconocimiento).

Finalmente, en el capítulo 5, exponemos nuestras conclusiones y esbozamos posibles líneas de trabajo futuro.

Capítulo 2

Detección de Rostros

La detección de rostros es actualmente utilizada en diversos tipos de aplicaciones. Por ejemplo, muchas cámaras fotográficas digitales, antes de tomar una fotografía, detectan los rostros presentes en la escena para mejorar el enfoque y la nitidez. También se utiliza la detección de rostros en aplicaciones de seguridad y seguimiento de personas, entre otras. En un sistema reconocedor de rostros es imprescindible, en primera instancia, detectar los rostros en una imagen para luego poder determinar la identidad de cada uno.

La detección de rostros determina la ubicación, tamaño y orientación de las caras que se encuentran presentes en una imagen. Cuando hablamos de ubicación, nos referimos a la posición de las caras en relación a la imagen. Además, las caras pueden presentarse con diversos tamaños, no sólo por las diferentes contexturas físicas de diferentes personas, sino también por la variedad de distancias de los rostros respecto a la cámara al momento de tomar la fotografía. En cuanto a la orientación, en este trabajo solamente trataremos con caras dispuestas frontalmente, aunque no se descartarán aquellas que presenten un leve cambio de orientación.

Existen diversos métodos para la detección de rostros. Sin embargo, muchos de ellos no son aplicables cuando se busca hacerlo en tiempo real. El método propuesto por Viola y Jones [46] fue el primero en ofrecer detección de rostros robusta y en tiempo real. Posee dos etapas principales: una de *entrenamiento* del detector, que es la que mayor tiempo de procesamiento demanda, y otra de *detección* donde se emplea el detector entrenado en la primera etapa sobre cada imagen a analizar. Esta segunda etapa es muy rápida y permite realizar la detección en tiempo real.

Para la detección de rostros, Viola y Jones analizan la imagen en busca de características relevantes que aporten información acerca de la presencia de una cara.

A continuación se mencionan algunas definiciones generales que más adelante formalizaremos.

Definición 2.1. *Un* ***feature*** f *de una imagen* I*, también llamado característica, contiene información de* I*, codificada tal que resulte relevante para llevar a cabo una tarea computacional dada.*

Un *feature* puede referirse, por ejemplo, al resultado de aplicar una operación en un área determinada de la imagen y su vecindad, o a una estructura o patrón dentro de la imagen, como por ejemplo: vértices, bordes, esquinas u otras estructuras más complejas. En el trabajo de Viola y Jones, se utiliza un algoritmo que busca, entre una gran cantidad y variedad de *features* de la imagen, aquellos que mayor información aportan acerca de una cara. Estos *features* pueden ser calculados muy eficientemente.

Una aplicación de los *features* es la de formar clasificadores. Un clasificador evalúa los *features* de una imagen de tal manera de poder decidir la presencia de un determinado objeto.

Definición 2.2. *Un* ***clasificador*** $c : I \to [1, K]$ *es una función que permite distinguir entre* K *clases de objetos dentro de una imagen* I*.*

En el trabajo de Viola y Jones [46], por ejemplo, se utilizan clasificadores para distinguir objetos que son cara de objetos que no lo son, es decir, $K = 2$. Construyen los clasificadores utilizando una variante de *features* conocidos como "Haar-like features" [36]. Estos se pueden ver en la figura 2.1. Están definidos por dos, tres o cuatro rectángulos y su cálculo consiste en hallar la diferencia entre la integral de la imagen en las zonas blancas y la integral de la imagen en las zonas negras (ver sección 2.1.1). Denominamos $\mathbb{F}$ al conjunto de tipos de *features* posibles.

$$\mathbb{F} = \left\{ F_1 : \quad , \; F_2 : \quad , \; F_3 : \quad , \; F_4 : \quad , \; F_5 : \quad \right\}$$

Figura 2.1: Tipos de *features* usados por Viola y Jones. F_1 y F_2: *features* compuestos por dos rectángulos. F_3 y F_4: *features* compuestos por tres rectángulos. F_5: *feature* de cuatro rectángulos.

A este tipo de *features*, se los llama *simple features* por ser muy sencillos de calcular. Aportan poca información en forma individual, pero poseen la característica de poder calcularse en forma extremadamente veloz, en orden constante, y se pueden combinar para formar clasificadores que brindan

mayor información. La etapa de entrenamiento descripta por Viola y Jones consiste justamente en esto, en formar clasificadores que sean capaces de discriminar entre caras y no caras, combinando otros clasificadores muy simples construidos en base a *simple features*.

Definición 2.3. *Un **clasificador débil** es un clasificador del cual se espera una performance muy baja, levemente mejor que el azar.*

Definición 2.4. *Un **clasificador fuerte** es un clasificador compuesto por varios clasificadores débiles y del que se espera un mejor desempeño, en comparación a un clasificador débil.*

Durante la etapa de entrenamiento, el método de Viola y Jones construye un clasificador fuerte utilizando una estrategia general, denominada *boosting*. A este proceso se lo conoce con el nombre de "aprender un clasificador". El *boosting* es un algoritmo de aprendizaje supervisado, es decir, un algoritmo que aprende una función a partir de un conjunto de muestras de entrenamiento que consisten en pares: la muestra y la salida esperada para esa muestra de entrada. La salida esperada es la respuesta que se obtendría de un clasificador ideal (aquel que, dada una muestra, siempre devuelve la clase correcta). El algoritmo de boosting más conocido es el *Adaboost*. Su nombre proviene de *Adaptative Boosting* y fue formulado por Yoav Freund y Robert Schapire en 1995[21]. Es adaptativo porque va ajustando los clasificadores débiles que se van agregando a un clasificador fuerte, en base a las muestras mal clasificadas por los clasificadores débiles ya agregados. En general, los algoritmos de boosting proceden de la siguiente manera:

1. Se tiene un conjunto de muestras observables $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, donde x_i es una muestra e y_i es la observación esperada para la muestra x_i, con $i : 1 \ldots N$.
2. Se asigna un peso inicial $p_{i,1}$ a cada muestra, para $i : 1 \ldots N$.
3. Dado T, la cantidad de clasificadores débiles a entrenar,

   Para $t = 1$ hasta T

   a) Con los pesos $p_{i,t}$ y las muestras, generar un clasificador débil WC_t llamando a un algoritmo que lo calcula. A este algoritmo se lo llama *weak learner* o algoritmo de aprendizaje débil.

   b) Calcular los pesos para la siguiente iteración $p_{i,t+1}$, para $i : 1 \ldots N$, de acuerdo a los datos obtenidos en el paso anterior.
4. Combinar los clasificadores débiles $WC_1 \ldots WC_T$ en un único clasificador fuerte SC utilizando un peso para cada clasificador débil de acuerdo a su efectividad.

La ventaja principal del *boosting* sobre otros métodos es la velocidad de aprendizaje y la capacidad de retroalimentación gracias a que posee, en cada iteración, información de las iteraciones anteriores representada en el peso de cada muestra de entrenamiento. Mediante los pesos se especifica la importancia de cada muestra en cada iteración.

En la siguiente sección, vemos una implementación específica del algoritmo de *boosting* aplicada al entrenamiento y detección de rostros realizada por Viola y Jones.

2.1. Algoritmos de Entrenamiento y Detección

En el trabajo de Viola y Jones [46] se presenta un método supervisado, muy robusto y rápido para la detección de rostros. Como mencionamos antes, el método posee dos etapas principales: una de *entrenamiento* del detector, donde se construyen los clasificadores que lo componen, y otra de *detección*, donde se emplea el detector entrenado en la primera etapa para detectar las caras en una imagen. Durante la etapa de entrenamiento se crean varios clasificadores fuertes o *strong classifiers* que permiten decidir cuáles regiones de la imagen corresponden a una cara y cuáles no. Luego se organizan los clasificadores fuertes en una estructura de cascada (ver figura 2.3) que permite mejorar la eficiencia del detector, dado que los primeros niveles de la cascada son clasificadores menos complejos y más veloces que los últimos. De esta manera se optimiza el proceso general de detección ya que la gran mayoría de los objetos que no son caras son filtrados en los niveles más rápidos.

El trabajo de Viola y Jones incluye tres contribuciones principales que veremos en las subsiguientes secciones. En primer lugar, se presenta una estructura novedosa denominada imagen integral que se utiliza para optimizar el cálculo de los *simple features*, tanto en el entrenamiento como en la detección. En segundo lugar, se presenta una forma de construir un clasificador fuerte utilizando una variante del algoritmo Adaboost. Este algoritmo se utiliza en la etapa de entrenamiento. En tercer lugar, se presenta la estructura de cascada mencionada anteriormente y un algoritmo para construirla durante la etapa de entrenamiento.

2.1.1. Imagen Integral

Viola y Jones introducen en su trabajo una representación de la imagen denominada imagen integral. La imagen integral permite calcular muy velozmente los *features* de la imagen utilizados tanto en el entrenamiento como en la detección. Los *features* son muy simples y se los denomina *simple features* (Ver definición 2.6).

Todas la imágenes utilizadas, tanto para el entrenamiento como para la detección, son imágenes en escala de grises.

Definición 2.5. *Una* ***imagen en escala de grises****, en dos dimensiones de tamaño $n \times m$, es una función $I : \mathbb{N}^2 \to \mathbb{N}$, donde $I(x, y) = i_{xy}$ es el valor o intensidad de la imagen en la posición (x, y), $1 \leq x \leq n$, $1 \leq y \leq m$.*

Definición 2.6. *Un* ***simple feature*** *o* ***haar-like feature*** *f, es una función definida por la siguiente ecuación:*

$$f_F = \sum_{r_i \in F} c_i \cdot RecSum\,(r_i) \quad , \quad \text{donde} \quad F \in \mathbb{F} \tag{2.1}$$

$$c_i = \begin{cases} -1 & si\ r_i\ es\ blanco \\ 1 & si\ r_i\ es\ negro \end{cases}$$

donde F es uno de los tipos de features mostrados en la figura 2.1, r_i son los rectángulos que componen al tipo F y RecSum es la función que calcula la suma de los píxeles de r_i.

Definición 2.7. *Dada una imagen I de tamaño $n \times m$, la imagen integral de I es una función II definida por la siguiente ecuación:*

$$II(x, y) = \sum_{\substack{1 \leq x' \leq x \\ 1 \leq y' \leq y}} I(x', y') \qquad para \qquad 1 \leq x \leq n\ ,\ 1 \leq y \leq m \tag{2.2}$$

Notación*: Dado un punto $p = (x_p, y_p)$ en la imagen, se escribe $II(p)$ para referirse a $II(x_p, y_p)$.*

La imagen integral puede calcularse en una sola pasada de la imagen original utilizando el siguiente par de ecuaciones recursivas:

$$S(x, y) = S(x, y - 1) + I(x, y) \tag{2.3}$$

$$II(x, y) = II(x - 1, y) + S(x, y) \tag{2.4}$$

Usando la imagen integral, la suma de los píxeles correspondientes a un rectángulo del *simple feature* puede calcularse con cuatro accesos a la matriz de

la imagen integral (ver figura 2.2). Esto implica que la diferencia entre dos rectángulos puede calcularse con solo ocho accesos, pero si los rectángulos son adyacentes y comparten un lado, el número de accesos se puede reducir a seis.

A modo de ejemplo, en la figura 2.2 se calcula la suma de los píxeles del

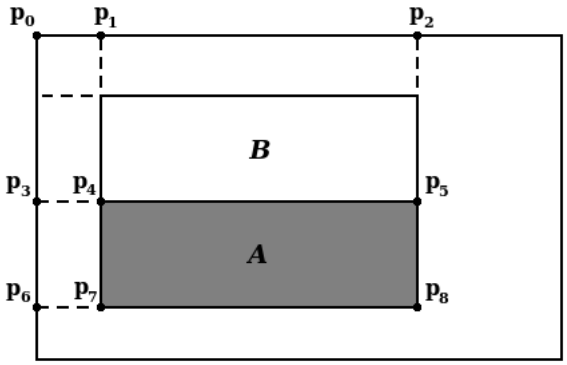


Figura 2.2: Cálculo de la suma de píxeles utilizando la imagen integral: $II(p_8)$ es la suma de los píxeles dentro del rectángulo formado por p_0, p_2, p_6 y p_8. $II(p_5)$ es la suma para el rectángulo dado por p_0, p_2, p_3 y p_5. $II(p_7)$ es la suma para el rectángulo dado por p_0, p_1, p_6 y p_7. $II(p_4)$ es la suma para el rectángulo dado por p_0, p_1, p_3 y p_4. La suma para el rectángulo A está dada por: $II(p_8) - II(p_5) - II(p_7) + II(p_4)$.

rectángulo A. El valor de la imagen integral en el punto p_8 ($II(p_8)$) es la suma de los píxeles dentro del rectángulo formado por p_0, p_2, p_6 y p_8. El valor en p_5 ($II(p_5)$) es la suma de los píxeles dentro del rectángulo formado por p_0, p_2, p_3 y p_5. El valor en p_7 ($II(p_7)$) es la suma de los píxeles dentro del rectángulo formado por p_0, p_1, p_6 y p_7. El valor en p_4 ($II(p_4)$) es la suma de los píxeles dentro del rectángulo formado por p_0, p_1, p_3 y p_4. Para calcular la suma de los píxeles bajo el rectángulo A, se realiza la siguiente operación: $II(p_8) - II(p_5) - II(p_7) + II(p_4)$, realizando solo cuatro accesos a la imagen integral.

La imagen integral, además, hace más rápida la detección de caras multiescala. Con esto nos referimos a la detección de caras de distintos tamaños dentro de la misma imagen. Para realizar la detección multiescala, otras implementaciones utilizan una estructura denominada pirámide de imágenes. La pirámide de imágenes se genera partiendo de la imagen a testear como base y generando varios niveles reescalando y aplicando filtros a la imagen del nivel anterior. Construir esta pirámide implica un procesamiento, previo a la detección, con alto costo computacional. En el trabajo de Viola y Jones se logra el mismo objetivo reescalando los *features* y utilizando una única imagen integral. Reescalar los *features* durante la deteccin es más rápido que

calcular la pirámide de imágenes completa al inicio de la detección.

La imagen integral se utiliza, además, para normalizar las imágenes de entrenamiento y la imagen de test en forma previa a la detección. De esta forma se reducen los efectos causados por diferencias de iluminación.
Para normalizar la imagen, se calcula la desviación estándar utilizando la siguiente ecuación:

$$\sigma^2 = m^2 - \frac{1}{N}\sum x^2 \tag{2.5}$$

donde σ es la desviación estándar, m es la media de la imagen y x es el valor de la imagen en un punto. El valor de la media m se puede obtener a partir de la imagen integral, mientras que el valor de $\sum x^2$ se obtiene de una segunda imagen integral que se construye de la siguiente manera:

$$II_{sqr}(x,y) = \sum_{x' \leq x, y' \leq y} \left(I\left(x', y'\right)\right)^2 \qquad \text{para} \qquad 1 \leq x, x' \leq n \ , \ 1 \leq y, y' \tag{2.6}$$

Es decir, que por cada imagen se generan dos imágenes integrales II e II_{sqr}. Una vez obtenida la desviación estándar σ de una imagen I, se puede normalizar I multiplicando cada píxel por el factor σ^{-1}. En vez de ello, Viola y Jones obtienen el mismo efecto de normalización, multiplicando por este mismo factor los valores de los *features* luego de evaluarlos.

2.1.2. Algoritmo de Boosting

En la primer etapa del algoritmo de Viola y Jones se entrena el detector. Para el entrenamiento se utiliza una gran cantidad de imágenes de caras y de no caras de tamaño fijo que se denominan *imágenes de entrenamiento*. El entrenamiento consiste en seleccionar entre un gran conjunto de *features*, aquellos que mejor representan una cara, y utilizarlos para formar clasificadores débiles y clasificadores fuertes. A estos features también se los denomina *features críticos*. Las imágenes de entrenamiento se utilizan para evaluar los features y determinar cuáles son los críticos.

El algoritmo de boosting que proponen Viola y Jones para el entrenamiento (ver algoritmo 2.1) es una variante del algoritmo de *Adaboost* y permite ir buscando los *features críticos* para formar clasificadores débiles y al mismo tiempo combinarlos para formar un clasificador fuerte que sea capaz de decidir la presencia de una cara. Se construyen T clasificadores débiles (uno por iteración). A cada clasificador débil también se lo llama hipótesis. La hipótesis final o clasificador fuerte, es una combinación lineal ponderada de las T hipótesis, en donde los pesos son inversamente proporcional a los erro-

res obtenidos en el entrenamiento.

El número total de posibles clasificadores débiles es extremadamente grande y esto hace muy difícil la selección de los mismos. Para seleccionar los *features*, el algoritmo de boosting utiliza otro algoritmo denominado *weak learner* o algoritmo de aprendizaje débil que es bastante agresivo y efectivo (ver algoritmo 2.2), y eso permite que se pueda definir como entrada al proceso de entrenamiento una gran cantidad de *features*. Sin embargo, el clasificador fuerte resultante es computacionalmente muy eficiente, ya que durante la detección sólo se evalúa una pequeña cantidad de features. Cada vez que se llama al *weak learner*, este aprende un clasificador débil que está compuesto por un único *feature*.

Definición 2.8. *Un clasificador débil está dado por la siguiente función:*

$$h(J, f, p, \theta) = \begin{cases} 1 & si \quad pf(J) < p\theta \\ 0 & en\ caso\ contrario \end{cases} \tag{2.7}$$

donde J es una subimagen o sección de la imagen de entrada I en la que se desea evaluar la presencia de una cara, f es el simple feature que compone al clasificador, $p \in \{-1, 1\}$ es la polaridad y θ es el umbral elegido.

Los clasificadores débiles pueden ser vistos como pequeños árboles de decisión de un único nodo. En la literatura, a este tipo de estructura, se la conoce también como *decision stumps*. El *weak learner* se limita a seleccionar aquellos clasificadores débiles que mejor separan las *muestras positivas* (aquellas que son caras) de las *muestras negativas* (aquellas que no lo son).

El algoritmo 2.1 se utiliza para entrenar un único clasificador fuerte en base a clasificadores débiles. Sin embargo, en vez de utilizar un único clasificador fuerte para la detección, Viola y Jones han introducido una estructura de varios clasificadores fuertes que permite invertir menor tiempo de procesamiento en zonas de la imagen que no contienen caras y mayor tiempo en zonas que si las poseen. En la siguiente sección se describe esta estructura y también se incluye el algoritmo que proponen para entrenar los clasificadores fuertes de acuerdo a la misma.

2.1.3. Cascada de Clasificadores Fuertes

Viola y Jones presentan un método para combinar los clasificadores fuertes en una estructura en forma de cascada que permite descartar rápidamente las regiones no importantes de la imagen denominadas *background*

Algoritmo 2.1 Algoritmo de boosting propuesto por Viola y Jones. Se construyen T hipótesis o clasificadores débiles. La hipótesis final o clasificador fuerte es una combinación lineal de las T hipótesis, cuyos pesos son inversamente proporcionales a los errores de entrenamiento.

1: $(x_1, y_1), \ldots, (x_n, y_n) \leftarrow$ muestras de entrenamiento, con $y_i = 0$ si la muestra es negativa (no es cara) e $y_i = 1$ si la muestra es positiva (es cara).

2: Inicializar los pesos:

$$\begin{cases} w_{1,i} = \frac{1}{2m} & \text{si} \quad y_i = 0 \\ w_{1,i} = \frac{1}{2l} & \text{si} \quad y_i = 1 \end{cases}$$

$$\begin{array}{rrll} \text{donde} & m & \leftarrow & \text{cantidad de imágenes negativas} \\ & l & \leftarrow & \text{cantidad de imágenes positivas} \end{array}$$

3: **for** $t = 1 \ldots T$ **do**

4: Normalizar los pesos: $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$

5: Seleccionar el clasificador débil de menor error, siendo el error:

$$\epsilon_t = \min_{f,p,\theta} \sum_i w_i \left| h\left(x_i, f, p, \theta\right) - y_i \right|$$

6: Se define h_t, el clasificador débil de la iteración t, como:

$$h_t\left(x\right) = h\left(x, f_t, p_t, \theta_t\right)$$

donde f_t, p_t y θ_t son los minimizadores de ϵ_t.

7: Actualizar lo pesos:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

$$\begin{array}{rll} \text{donde} & e_i = 0 & \text{si } x_i \text{ fue bien clasificada} \\ & e_i = 1 & \text{si } x_i \text{ fue mal clasificada} \\ & \beta_t = \frac{\epsilon_t}{1-\epsilon_t} & \end{array}$$

8: **end for**

9: El clasificador fuerte final está dado por:

$$C(x) = \begin{cases} 1 \quad \text{si} \quad \sum_{t=1}^{T} \alpha_t h_t\left(x\right) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 \qquad \text{en caso contrario} \end{cases}$$

$$\text{donde} \quad \alpha_t = \log \frac{1}{\beta_t}$$

Algoritmo 2.2 Algoritmo de selección de clasificadores débiles (weak learner)

selectedFeature $\leftarrow \emptyset$
selectedFeatureError $\leftarrow \infty$
$T^+ \leftarrow$ la suma total de los pesos de las muestras positivas
$T^- \leftarrow$ la suma total de los pesos de las muestras negativas
for all $f \in features$ **do**
  $X^{(f)} \leftarrow$ muestras de entrenamiento ordenadas por el valor de f
  $e_f \leftarrow \infty$
  for $i = 1$ to N **do**
    $S_i^+ \leftarrow$ la suma de los pesos de las muestras positivas anteriores a $x_i^{(f)}$
    $S_i^- \leftarrow$ la suma de los pesos de las muestras negativas anteriores a $x_i^{(f)}$
    $e_i \leftarrow \text{mín}(S_i^+ + (T^- - S_i^-), S_i^- + (T^+ - S_i^+))$
    $e_f \leftarrow \text{mín}(e_f, e_i)$
  end for
  if selectedFeatureError $> e_f$ **then**
    selectedFeature $\leftarrow f$
    selectedFeatureError $\leftarrow e_f$
  end if
end for

(todo aquello que no es cara) e invertir la mayor parte del tiempo de procesamiento en aquellas zonas que posiblemente contengan una cara. Esto no sólo reduce el tiempo computacional sino también mejora la precisión de la detección. El esquema de la cascada propuesta se puede ver en la figura 2.3. Allí se observa una cascada de N etapas. Cada etapa está compuesta por un único clasificador fuerte. En caso de que, en una etapa, una subimagen sea detectada como *background* o no cara, se descarta esa subimagen. En caso de que sea detectada como cara, pasa a ser evaluada por la siguiente etapa. En la última etapa, las subimágenes detectadas como caras son el resultado final del algoritmo de detección. Las primeras etapas deben evaluar mayor cantidad de imágenes y por ello se diseñan los primeros clasificadores fuertes para que tengan un costo computacional mucho menor que los de las últimas etapas.

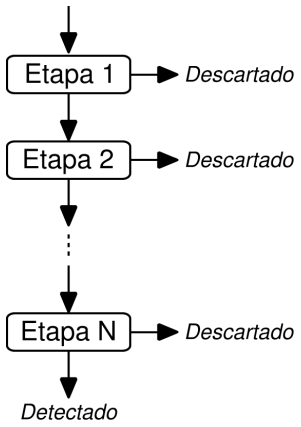


Figura 2.3: Cascada de clasificadores fuertes propuesta por Viola y Jones

Para diseñar una cascada se plantean algunos objetivos de performance. El diseño y la complejidad de la cascada depende de estos. Uno de ellos, por ejemplo, es la *tasa de falsos positivos* (FPR - False Positive Rate) buscada. Este índice mide la cantidad de objetos que no son caras y que son detectados como si lo fueran. Otros índices de interés son: la *tasa de falsos negativos* (FNR - False Negative Rate), que mide la cantidad de caras que no son detectadas como tal, y la *tasa de detección* (DR - Detection Rate), que mide el porcentaje de detecciones respecto al total de caras.

En una cascada de K etapas, el FPR total F de la cascada, y el DR total D de la cascada están dados por:

$$F = \prod_{i=1}^{K} f_i \tag{2.8}$$

$$D = \prod_{i=1}^{K} d_i \tag{2.9}$$

donde f_i y d_i es el FPR y el DR de cada etapa de la cascada respectivamente, es decir, de cada clasificador fuerte.

En base a estos índices, Viola y Jones plantean en [46] el algoritmo 2.3 que se utiliza para el entrenamiento de la cascada. El mismo recibe como parámetros: el máximo FPR aceptable por etapa (f), el mínimo DR aceptable por etapa (d), el FPR general de la cascada (F_{target}), el conjunto de muestras positivas (P), el conjunto de muestras negativas (N) y el conjunto de muestras de validación (V), compuesto por muestras positivas y negativas.

2.1.4. Algoritmo de Detección

Una vez entrenada la cascada de clasificadores, se utiliza como parte del algoritmo de detección. Para detectar las caras en una imagen, se realiza una exploración exhaustiva de la misma, evaluando la cascada de clasificadores en muchas posiciones y a diferentes escalas para contemplar distintos tamaños de caras (Ver algoritmo 2.4).

Las posiciones en las que se evalúa el detector varían de acuerdo a un Δ de píxeles seleccionado, que determinan el "paso" del detector. El Δ afecta tanto la velocidad del detector como su precisión. Cuanto mayor es Δ mayor es la velocidad del detector y menor su precisión.

En contraste con otras técnicas de detección, el escalado se realiza sobre los clasificadores de la cascada en vez de escalar la imagen a testear. El escalado de los clasificadores se logra modificando el tamaño de los *features* que los componen al momento de evaluarlos por un factor C ingresado como parámetro. Para calcular los *features* se utiliza la misma imagen integral independientemente de la escala, por lo que el costo computacional de escalar los *features* y de evaluarlos es ínfimo.

En la siguiente sección se presentan una serie de mejoras realizadas sobre

Algoritmo 2.3 Algoritmo para entrenar una cascada de clasificadores propuesto por Viola y Jones

1: $f \leftarrow$ máximo FPR admitido por etapa
2: $d \leftarrow$ mínimo DR admitido por etapa
3: $F_{target} \leftarrow$ FPR general de la cascada admitido
4: $P, N, V \leftarrow$ conjuntos de muestras positivas, negativas y de validación
5: $F_0 \leftarrow 1$
6: $D_0 \leftarrow 1$
7: $i \leftarrow 0$
8: **while** $F_i > F_{target}$ **do**
9: $i = i + 1$
10: $n_i = 0$
11: $F_i = F_{i-1}$
12: **while** $F_i > f \times F_{i-1}$ **do**
13: $n_i = n_i + 1$
/* Entrenar un clasificador fuerte utilizando N, P y n_i cantidad de features: */
14: $TrainClassifier(N, P, n_i)$
/* Evaluar la cascada de clasificadores utilizando el conjunto de muestras de validación V y actualizar F_i y D_i : */
15: $F_i, D_i \leftarrow EvaluateCascade(V)$
/* Reducir el umbral del clasificador i hasta que para la cascada actual se cumpla: $DR \geq d \times D_{i-1}$. tener en cuenta que este ajuste afecta a F_i : */
16: $AdjustThreshold(C_i)$
17: **end while**
18: **if** $F_i > F_{target}$ **then**
19: /* Asignar a N los falsos positivos detectados por la cascada actual: */
20: $N \leftarrow FalseDetections(N)$
21: **end if**
22: **end while**

Algoritmo 2.4 Algoritmo de detección para una cascada entrenada

1: $window\_size \leftarrow window\_size_0$ /* $window$ es la ventana en la que se evalúan los features. $window\_size_0$ es el tamaño de las imágenes de entrenamiento */
2: $scale \leftarrow 1$
3: $faces \leftarrow \emptyset$
4: **while** $window\_size \leq image\_size$ **do**
5: $window\_size \leftarrow window\_size$ escalada por $scale$
6: $cascade \leftarrow cascade_0$ con los features escalados por $scale$ /* $cascade_0$ es la cascada entrenada */
7: $dX \leftarrow \Delta \times scale$ /* Δ constante de incremento de posición */
8: $dY \leftarrow \Delta \times scale$
9: $Y \leftarrow 0$
10: **while** $Y \leq image\_height - window\_height$ **do**
11: $X \leftarrow 0$
12: **while** $X \leq image\_width - window\_width$ **do**
13: **if** $test\_cascade(X, Y)$ es cara **then**
14: $faces \leftarrow faces \cup \{X; Y\}$
15: **end if**
16: $X \leftarrow X + dX$
17: **end while**
18: $Y \leftarrow Y + dY$
19: **end while**
20: $scale \leftarrow scale \times C$ /* C constante de incremento de escala */
21: **end while**

el algoritmo de entrenamiento propuesto originalmente por Viola y Jones, y que se han ido publicando en diferentes trabajos realizados sobre la detección de objetos.

2.2. Mejoras sobre los Algoritmos de Entrenamiento y Detección

En la presente sección se presentan y analizan avances, realizados por diferentes autores, que mejoran el método de detección de caras visto en la sección 2.1.

En las secciones 2.2.1 y 2.2.2, citamos el trabajo publicado por Rainer Lienhart y Jochen Maydt [31], donde se extiende el algoritmo de Viola y Jones a través de dos contribuciones principales: un conjunto de *features* extendido y un algoritmo de post-optimización para una cascada de clasificadores.

Luego, en la sección 2.2.3, se presenta el algoritmo de Gentle Boost [22], con el que Lienhart, Kuranov y Pisarevsky [30] han realizado experimentos, obteniendo mejores resultados que con otros algoritmos de boosting.

En el mismo trabajo, se demuestra en forma empírica la superioridad de los CART (*Classification And Regression Trees*) utilizados como clasificadores débiles en lugar de los de un solo nodo empleados por Viola y Jones. En la sección 2.2.4, se introducen los CART.

En un trabajo posterior, Lienhart, Lian y Kuranov [29] proponen una estructura de clasificadores nueva, que amplía la cascada propuesta inicialmente, formando un árbol de clasificadores. Esta estructura la presentamos en la sección 2.2.5. Resulta ser más eficiente que una cascada, ya que permite que las etapas se dividan en diversas ramas para tratar en forma diferente con las distintas clases de objetos que puedan presentarse en una imagen.

Finalmente, se presenta una mejora propuesta por Lienhart, Kuranov y Pisarevsky [30] que se aplica en el algoritmo de detección al momento de escalar los *features*.

2.2.1. Conjunto de Features Extendido

En primer lugar, Lienhart y Maydt, amplían el conjunto original de *features*, agregando nuevos *features* rotados a 45°. Para que la evaluación de

los *features* rotados sea eficiente, proponen una representación análoga a la imagen integral. En la publicación se menciona que, al utilizar estos nuevos *features*, se obtiene para una tasa de detección (DR) dada, un 10 % menos de falsos positivos (FP). El conjunto nuevo de *features* se puede observar en la figura 2.4. El *feature* 2.4(ñ), no es utilizado por Lienhart y Maydt, dado que el mismo puede ser aproximado por los *features* 2.4(i) y 2.4(k).

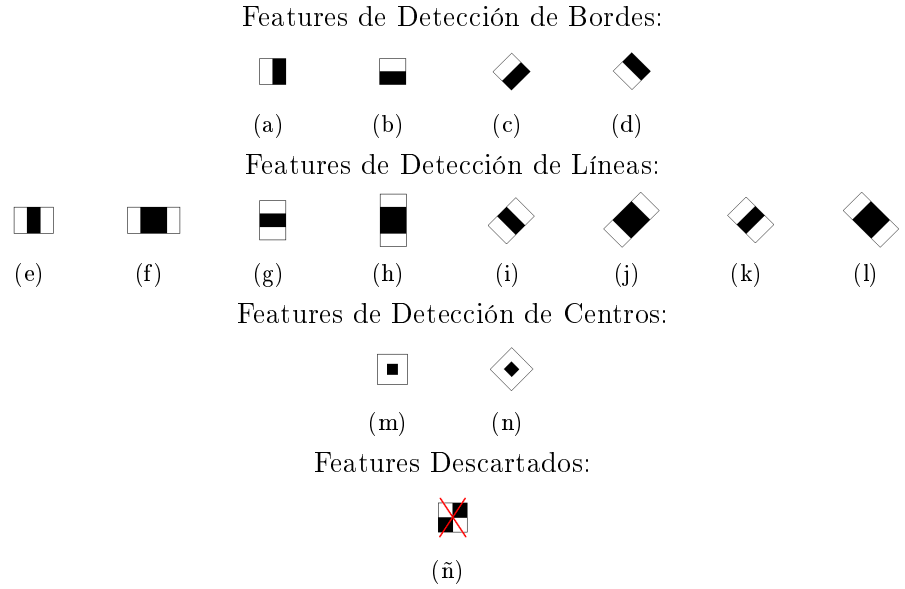


Figura 2.4: Features extendidos, utilizados por Lienhart y Maydt.

Para calcular los *features* sin inclinación: 2.4(a), (b), (e), (f), (g), (h) y (m), utilizan una función denominada SAT (*Summed Area Table*), que es similar a la imagen integral propuesta por Viola y Jones. Como vimos antes, puede calcularse en una sola pasada por los píxeles de la imagen y está dada por:

$$SAT(x,y) = \sum_{\substack{1 \le x' \le x \\ 1 \le y' \le y}} I(x',y') \tag{2.10}$$

Además agregan la función RSAT (*Rotated Summed Area Table*), que se utiliza para calcular los *features* a 45°: 2.4(c), (d), (i), (j), (k), (l) y (n). Puede calcularse con dos pasadas por los píxeles de la imagen y está dada por:

$$RSAT(x,y) = \sum_{\substack{1 \leq x' \leq x \\ x' \leq x - |y - y'|}} I(x', y') \tag{2.11}$$

2.2.2. Post-Optimización de Clasificadores Fuertes

Como segunda contribución, Lienhart y Maydt proponen un procedimiento de post-optimización para una cascada, que permite mejorar el FPR de la misma un 12.5 % en promedio. El mismo consiste en optimizar cada clasificador fuerte de la cascada, ajustando los umbrales de los *features* que forman el clasificador y buscando bajar el FPR pero sin perder el DR para el cual fue entrenado. De esta manera se logra reducir el error total del clasificador (FPR + FNR). Dado el clasificador fuerte:

$$C(x) = signo\left(\sum_{t=1}^{T} \alpha_t \cdot h(x, f_t, p_t, \theta_t) + b\right) \tag{2.12}$$

los parámetros libres son los umbrales θ_t y b, dado que los α_t deben ser elegidos de acuerdo al algoritmo de Adaboost para preservar sus propiedades. Empezando por el θ_t original (el encontrado por el *weak learner*), se va incrementando y decrementando su valor en forma de gradiente descendente, verificando su performance. Cabe mencionar que no se puede implementar un verdadero gradiente descendente dado que $C(x)$ no es una función continua. Cualquier cambio en θ_t requiere recalcular α_k y w_{k+1} $\forall k \geq t$. El procedimiento de post-optimización es iterativo y se describe en el algoritmo 2.5.

2.2.3. Gentle Adaboost

En el trabajo publicado por Lienhart, Kuranov y Pisarevsky [30], se exponen diversos experimentos realizados sobre tres variantes del algoritmo de Adaboost, a saber: *Discrete Adaboost* (también conocido como *Adaboost.M1* [20]), *Real Adaboost* [22] y *Gentle Adaboost* [22]. El algoritmo propuesto por Viola y Jones en la sección 2.1 es una variante de *Discrete Adaboost*. Lienhart, Kuranov y Pisarevsky muestran empíricamente que *Gentle Adaboost* es más efectivo que *Discrete* y *Real Adaboost*, requiriendo una menor cantidad de *features* para obtener el mismo performance, y reduciendo el tiempo de cómputo.
En [22] se muestra que *Discrete Adaboost* sigue un modelo aditivo de regresión logística (*additive logistic regression model*) que minimiza la esperanza de la función de pérdida $e^{-y \cdot C(x)}$, con $C(x)$ el clasificador entrenado. Cada clasificador produce una "etiqueta" para la muestra evaluada indicando como

Algoritmo 2.5 Algoritmo de post-optimización, propuesto por Lienhart y Maydt, para un clasificador y una tasa de detección dada.

1: Se define $H_T = \sum_{t=1}^{T} \alpha_t \cdot h\left(x, f_t, p_t, \theta_t\right)$ y $H_T^j = \sum_{t=1, t\neq j}^{T} \alpha_t \cdot h\left(x, f_t, p_t, \theta_t\right)$

2: Sean:

$$\begin{array}{rcl} \left(x_1^P, y_1^P\right), \ldots, \left(x_n^P, y_n^P\right) & \rightarrow & \text{muestras positivas} \\ \left(x_1^N, y_1^N\right), \ldots, \left(x_m^N, y_m^N\right) & \rightarrow & \text{muestras negativas} \\ C(x) = signo\left(H_T + b\right) & \rightarrow & \text{el clasificador a optimizar con } b \text{ sujeto a} \\ & & E_w^P\left[1_{y_i^P = C\left(x_i^P\right)}\right] / E_w^P[1] \geq d \\ d & \rightarrow & \text{la tasa de detección (DR) buscada para} \\ & & \text{cada etapa de la cascada} \end{array}$$

donde E_w^P y E_w^N denotan la esperanza de los pesos w calculada en base a muestras positivas y negativas respectivamente.

3: Inicializar:

$$\begin{array}{rcl} err & \leftarrow & E_w^N\left[1_{y_i^N \neq C\left(x_i^N\right)}\right] \\ errOld & \leftarrow & err + 1 \end{array}$$

4: **while** $err < errOld$ **do**

5: $errOld = err$

6: **for** $j = 1$ to T **do**

7: Encontrar la combinación $\{\theta_j, b\}$ que minimiza el FPR esperado, sujeto a la tasa de detección d:

$$\left\{\tilde{\theta}_j, \tilde{b}_j\right\} \leftarrow \arg\min_{\theta_j, b}\ err^N\left(j, \theta_j, b\right)$$

donde

$$err^N\left(j, \theta_j, b\right) = E_w^N\left[1_{y_i^N \neq \text{sign}\left(H_T^j + \alpha_j \cdot h\left(x_i^N, f_j, p_j, \theta_j\right) + b\right)}\right] / E_w^N[1]$$

sujeto a

$$E_w^P\left[1_{y_i^P \neq \text{sign}\left(H_T^j + \alpha_j \cdot h\left(x_i^P, f_j, p_j, \theta_j\right) + b\right)}\right] / E_w^P[1] \geq d$$

y

α_j y $w_{j,i}$ mantienen los valores asignados por Adaboost,

E^P denota la esperanza calculada en base a las muestras positivas,

err^N y E^N denotan el error y la esperanza en base a las muestras negativas.

8: **end for**

9: Determinar la tupla $\left\{\tilde{\theta}_j, \tilde{b}_j\right\}$ con menor FPR esperado, sujeto a d:

$$\tilde{j} = \arg\min_j \left(err^N\left(j, \tilde{\theta}_j, \tilde{b}_j\right)\right)$$

10: Asignar:

$$\begin{aligned} \theta_{\tilde{j}} &\leftarrow \tilde{\theta}_{\tilde{j}} \\ b &\leftarrow \tilde{b}_{\tilde{j}} \\ \alpha_j \ , \ w_{\theta,j} &\leftarrow \text{actualizar según Adaboost} \\ err &\leftarrow err^N\left(\tilde{j}, \tilde{\theta}_{\tilde{j}}, \tilde{b}_{\tilde{j}}\right) \end{aligned}$$

11: **end while**

única información si es de una clase u otra. Se propone entonces el algoritmo *Real Adaboost* que obtiene como predicción un número real. Este valor se usa para representar cuánto contribuye una observación en el modelo final. Tanto *Discrete Adaboost* como *Real Adaboost* intentan minimizar la esperanza de la función de pérdida $e^{-y \cdot C(x)}$. Luego se propone el algoritmo *Gentle Adaboost*, que minimiza la función de pérdida aplicando Newton.

En las figuras 2.6 y 2.7 pueden verse las principales diferencias entre *Discrete Adaboost* y *Gentle Adaboost*.

Algoritmo 2.6 Discrete Adaboost

1: Dadas n muestras de entrenamiento:
$(x_1, y_1), \ldots, (x_n, y_n)$, $y_i \in \{-1, 1\}$
2: Inicializar los pesos:
$w_{1,i} = 1/n$, $i = 1 \ldots n$
3: **for** $t = 1 \ldots T$ **do**
4: Formar el clasificador $h_t(x) \in \{-1, 1\}$ usando los pesos w_i en los datos de entrenamiento
5: Calcular $err_t = E_w\left[1_{y \neq h_t(x)}\right]$ y $\alpha_t = \log\left(\frac{1-err_t}{err_t}\right)$
6: Asignar $w_i \leftarrow w_i \cdot \exp\left(\alpha_t \cdot 1_{y_i \neq h_t(x_i)}\right)$, $i = 1 \ldots n$ y renormalizar los pesos tal que $\sum_i w_i = 1$
7: **end for**
8: Devolver el clasificador: $sign\left[\sum_{t=1}^{T} \alpha_t \cdot h_t(x)\right]$

Algoritmo 2.7 Gentle Adaboost

1: Dadas n muestras de entrenamiento:
$(x_1, y_1), \ldots, (x_n, y_n)$, $y_i \in \{-1, 1\}$
2: Inicializar los pesos:
$w_{1,i} = 1/n$, $i = 1 \ldots n$
3: **for** $t = 1 \ldots T$ **do**
4: Formar la función de regresión $h_t(x) \in \{-1, 1\}$ usando cuadrados mínimos de y_i a x_i con los pesos w_i en los datos de entrenamiento
5: Actualizar $C(x) \leftarrow C(x) + h_t$
6: Actualizar $w_i \leftarrow w_i \cdot \exp(-y_i \cdot h_t(x_i))$, $i = 1 \ldots n$ y renormalizar los pesos tal que $\sum_i w_i = 1$
7: **end for**
8: Devolver el clasificador: $sign[C(x)] = sign\left[\sum_{t=1}^{T} h_t(x)\right]$

2.2.4. Stump vs CART

Lienhart, Kuranov y Pisarevsky [30], realizaron pruebas para comprobar la eficacia de los clasificadores débiles. En ellas muestran la superioridad de los clasificadores que emplean los *features* extendidos (ver figura 2.4) sobre los propuestos originalmente por Viola y Jones (ver figura 2.1). Con el conjunto de *features* extendidos, se obtienen en promedio, un 10 % menos de falsos positivos. Por otro lado, se demuestra, en forma empírica, que la detección mejora considerablemente cuando se utilizan como clasificadores débiles pequeños árboles de decisión denominados CART (*Classification and Regression Trees* [8] [34]) en lugar de los clasificadores débiles o árboles de un solo nodo utilizados por Viola y Jones, también denominados *stumps*. Las velocidades de detección que se obtienen son similares, dado que con una mayor cantidad de nodos en los CARTS se requieren menos clasificadores débiles para alcanzar el performance requerido en cada etapa. Según los autores, los clasificadores débiles compuestos por *stumps*, no permiten aprender las dependencias existentes entre distintos *features*.

Los CART, también conocidos como árboles de decisión binarios, son árboles que fueron concebidos y optimizados para la toma de decisiones clínicas. Son binarios porque en cada nodo, el grupo de datos representado puede ser partido únicamente en dos subgrupos a través de una pregunta simple o la evaluación de un *simple feature*. Son recursivos porque el proceso de particionado se aplica una y otra vez hasta alcanzar uno de los criterio de corte que mencionamos más adelante.

El algoritmo define una métrica de impureza que se evalúa sobre los datos

de un nodo del árbol. La impureza es mínima cuando los datos del nodo pertenecen todos a la misma clase representada por el nodo. Utilizando esta métrica, el procedimiento de construcción del árbol busca qué *simple feature* combinado con qué umbral "purifican" o particionan mejor los datos para un nodo en particular. Luego, se repite el procedimiento para cada rama del árbol, hasta que los datos son suficientemente puros, o se alcanza un mínimo de datos por nodo o se alcanza un máximo de particiones impuesto.

Para la impureza se suelen utilizar tres métricas conocidas como: *Entropy Impurity*, *Gini Impurity* y *Misclassification Impurity*.

$$\textbf{Entropy Impurity:} \qquad imp\left(\theta\right) = -\sum_{j} P\left(\varphi_j\right) \cdot \log P\left(\varphi_j\right) \tag{2.13}$$

$$\textbf{Gini Impurity:} \qquad imp\left(\theta\right) = \sum_{j \neq i} P\left(\varphi_i\right) \cdot P\left(\varphi_j\right) \tag{2.14}$$

$$\textbf{Misclassification Impurity:} \qquad imp\left(\theta\right) = 1 - \max P\left(\varphi_j\right) \tag{2.15}$$

donde $P\left(\varphi_i\right)$ es la fracción de objetos pertenecientes a la clase φ_i incluidos en el nodo θ.

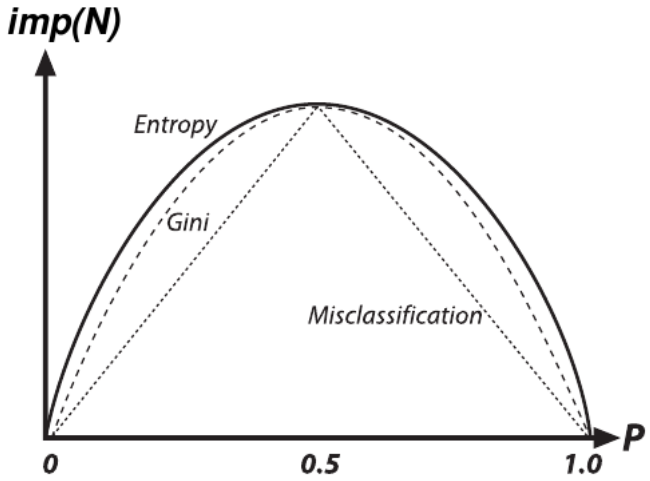


Figura 2.5: Métricas de Impureza

2.2.5. Árbol de Clasificadores Fuertes

En un trabajo posterior, Lienhart, Lian y Kuranov [29] presentan un detector compuesto por clasificadores fuertes, similar al propuesto por Viola y Jones, pero en el que los clasificadores se organizan en una estructura de árbol en vez de cascada. Cada nodo del árbol posee asociado un clasificador fuerte, de tal forma que, si el árbol posee una única hoja, la estructura equivale a la cascada utilizada por Viola y Jones. La estructura de árbol permite que las etapas de una cascada se dividan en diversas ramas para tratar en forma diferente con las distintas clases de objetos que puedan presentarse en una imagen. Para el entrenamiento de cada clasificador se emplea un algoritmo de boosting, pero se agrega un paso, denominado *Clustering And Splitting*, que permite construir las bifurcaciones en los nodos recursivamente. En el paso de *Clustering And Splitting* se decide realizar una bifurcación si y sólo si la división mejora el poder discriminatorio de un clasificador monolítico y tiene un menor costo computacional.

El motivo que los llevó a pensar en la nueva estructura de árbol es que los clasificadores suelen sobrecargarse para aprender muchas clases de objetos a la vez. Intuitivamente, se podría pensar en generar varias cascadas, cada una dedicada a un conjunto de clases más homogéneo. Sin embargo, esta solución no contrarresta dos dificultades. Por un lado, en la práctica es muy difícil agrupar los atributos o características de los objetos en sub-clases debido a ambigüedades o sub-clases que no son disjuntas, y por el otro, el hecho de tener que evaluar múltiples clasificadores especializados aumenta considerablemente el costo computacional, perdiendo la posibilidad de realizar la detección en tiempo real. A la estructura de árbol se la puede ver como una estructura de múltiples cascadas que combina las primeras etapas de cada una de ellas, preservando la precisión y reduciendo el costo computacional del detector (estrategia *Divide and Conquer*). En la figura 2.6 se comparan las tres estructuras: cascada, múltiples cascadas y árbol.

En la nueva estructura, se genera un clasificador fuerte por cada nodo del árbol. El entrenamiento comienza por el nodo raíz. Este nodo se distingue del resto por no poseer nodo padre. El clasificador del nodo raíz se entrena utilizando el total de muestras positivas más la cantidad de muestras negativas especificadas para cada etapa. Para el resto de los nodos, en cambio, se utilizan solo las muestras clasificadas como positivas por su correspondiente nodo padre (*True Positives* + *False Positives*). Los TP (*True Positives*) del nodo padre se usan como muestras positivas y los FP (*False Positives*) del nodo padre se usan como muestras negativas en el entrenamiento de el o los nodos hijos.

Para cada nodo del árbol se realiza un entrenamiento que obtiene como

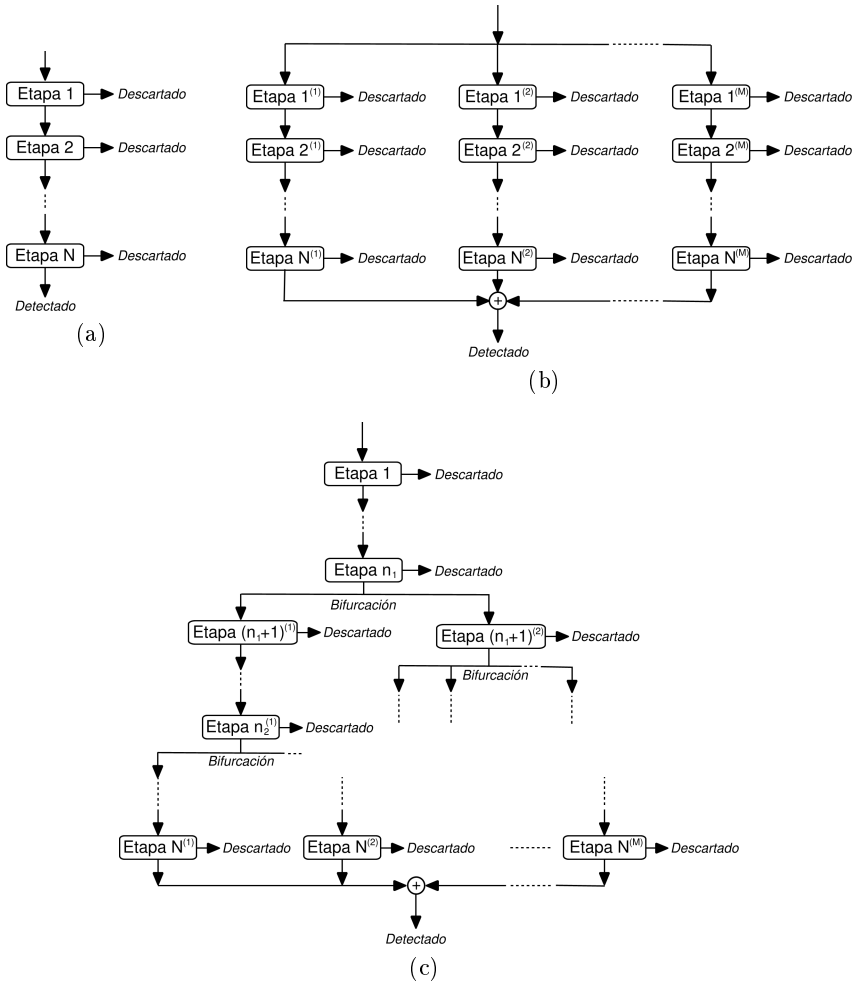


Figura 2.6: Diferentes estructuras de clasificadores fuertes. **(a)** Estructura de cascada propuesta por Viola y Jones. **(b)** Estructura de múltiples cascadas. **(c)** Estructura de árbol propuesta por Lienhart y Maydt.

resultado un clasificador fuerte cuya complejidad computacional, al momento de evaluarlo durante la detección, es lineal en el número de clasificadores débiles que lo componen. Por lo tanto, se busca que un clasificador fuerte esté formado por la menor cantidad posible de *features*, siempre que se alcance el DR (*Detection Rate*) y FPR (*False Positive Rate*) buscados.

El algoritmo de entrenamiento del árbol se puede ver más en detalle en el algoritmo 2.8.

Además de entrenar un único clasificador fuerte, se dividen las muestras en k grupos o clases, con $k \geq 2$ y $k < K_{max}$ para un K_{max} dado, y se en-

trenan k clasificadores fuertes que, en conjunto, cumplen con el DR y FPR buscados (*Divide and Conquer*). El criterio utilizado para decidir quedarse con un único clasificador fuerte o con k clasificadores fuertes es la opción que posee menor número total de *features*.

Para entrenar k clasificadores, se emplea un algoritmo denominado *k-means* [33], que agrupa las muestras positivas bien clasificadas por el nodo padre en k grupos o clases distintas. Luego, se utiliza cada grupo de muestras positivas y las muestras negativas mal clasificadas por el nodo padre para entrenar un clasificador.

Algoritmo 2.8 Algoritmo de entrenamiento del árbol de clasificadores propuesto por Lienhart, Lian y Kuranov.

```
1. struct TreeNode
2.    TreeNode next
3.    TreeNode child
4.    TreeNode parent
5.    TrainingData positiveSamples
6.    Int evaluate(sample) /* Evalúa la muestra (sample) en la cascada que
      va desde el nodo raíz hasta el nodo actual */
7. end struct
```

```
1. procedure startTreeTraining()
2.    TreeNode tn
3.    tn.parent ← 0
4.    tn.positiveSamples ← todas las muestras positivas
5.    tn.next ← 0
6.    nodeTraining(tn, 0, TARGET_HEIGHT_OF_TREE)
7. end procedure
```

```
1. procedure nodeTraining (TreeNode parent, Int curLevel, Int
   stopLevel)
2.    if (stopLevel = curLevel) then
3.       Return
4.    end if
5.    SPOS ← parent.evaluate(parent.positiveSamples) /* Filtrar las
      muestras positivas asignadas al nodo padre */
6.    SNEG ← parent.evaluate(negativeSamples) /* Filtrar #CNEG
      muestras negativas */
7.    S^1 ← trainStageClassifier(SPOS + SNEG)
```

8. BestClassifier $\leftarrow S^1$
9. BestNumberOfFeatures $\leftarrow O(S^1)$ /* $O(S^1)$ el nro. de *features* de S^1 */
10. **for** $k = 2$ to K_{max} **do**
11. Calcular en las muestras $SPOS$ los *features* usados por S^1.
12. Emplear el algoritmo *k-means clustering* sobre los datos de los *features* para agrupar las muestras positivas en k grupos $SPOS_i$ de muestras positivas.
13. Entrenar k clasificadores S_i^k con $SPOS_i$ y $SNEG$
14. **if** (BestNumberOfFeatures $> O(S_1^k) + \ldots + O(S_k^k)$) **then**
15. BestNumberOfFeatures $\leftarrow O(S_1^k) + \ldots + O(S_k^k)$
16. BestClassifier $\leftarrow \left\{S_1^k, \ldots, S_k^k\right\}$
17. **end if**
18. **end for**
19. TreeNode $\text{tn}_0 \leftarrow 0$
20. **for all** $S_i^k \in$ BestClassifier **do**
21. TreeNode tn_i
22. tn_i.parent $\leftarrow$ parent
23. tn_i.positiveSamples $\leftarrow \text{SPOS}_\text{i}$
24. tn_i.next $\leftarrow \text{tn}_{\text{i}-1}$
25. **end for**
26. **end procedure**

2.2.6. Escalado de Features en la Detección

En el trabajo de Lienhart, Kuranov y Pisarevsky [30] se presenta una mejora en el escalado de los *features* que se realiza durante el barrido de la imagen en la etapa de detección. Al escalar los *features*, la relación entre las áreas que componen al mismo puede cambiar por errores de redondeo. Como mejora, se agregan pesos a cada área del *feature* para que se mantenga la relación original utilizada durante el entrenamiento. De esta forma, el valor de un *feature* calculado en la ecuación 2.6 pasa a calcularse según la ecuación 2.16. Los resultados que obtuvieron al implementar esta solución muestran una mejora importante.

$$f_F = \sum_{r_i \in F} w_{r_i} \cdot c_i \cdot RecSum\left(r_i\right) \tag{2.16}$$

donde w_{r_i} es el peso asociado al rectángulo r_i y $RecSum\left(r_i\right)$ es la función que calcula la suma de los píxeles de r_i utilizando las funciones SAT y $RSAT$ vistas anteriormente.

2.3. Consideraciones Generales

En este capítulo abordamos la detección de rostros en una imagen a partir de un algoritmo robusto que permite obtener resultados en tiempo real. Consiste de dos etapas principales. En la primera, se utiliza *boosting* para entrenar el detector y, en la segunda, se recorre la imagen evaluando el detector entrenado en busca de caras. El *boosting* es una herramienta muy poderosa de aprendizaje. Combina un conjunto de clasificadores débiles, cuyo performance es levemente mejor que el azar, en un poderoso comité de votación o clasificador fuerte, cuyo performance muchas veces supera a soluciones como SVMs (*Support Vector Machines*) o redes neuronales.

El proceso de entrenamiento es la parte más crítica de la detección, dado que es en esta etapa que se define qué tipo de caras podrá encontrar el detector. De acuerdo a la aplicación que se le de al detector, el entrenamiento puede ser guiado de diferentes maneras. Por ejemplo, en nuestra aplicación de reconocimiento de rostros, nos interesa reducir los falsos positivos detectados y las caras que no estén en disposición frontal, como así también nos interesa maximizar la detección de caras ocluidas parcialmente por anteojos de sol, bufandas y cabellos. Para ello es fundamental seleccionar cuidadosamente las imágenes que se utilizan en el entrenamiento, las cuales se dividen en dos grupos. Un grupo de imágenes de caras, que tiene el objetivo de enseñarle al detector qué cosas son cara (incluyendo caras ocluidas), y un grupo de

imágenes sin caras, que tiene el objetivo de enseñarle al detector qué cosas no son cara. El objetivo final del entrenamiento es encontrar una función, o clasificador, que distinga correctamente entre ambos grupos de imágenes.

En el comienzo del capítulo presentamos el algoritmo de entrenamiento y detección propuesto por Viola y Jones. Luego fuimos mencionando las mejoras introducidas por diferentes autores. La primera de ellas consiste en un conjunto de *features* extendidos, entre los que se agregan *features* rotados a 45°, que permiten obtener un mejor conocimiento del dominio de imágenes. Junto con el conjunto de *features* se extiende el eficiente método de cálculo propuesto por Viola y Jones para contemplar los nuevos *features* y no perder velocidad de cómputo. Luego se presenta un algoritmo de post-optimización, que se aplica a cada clasificador fuerte entrenado en la cascada de clasificadores, y que ajusta los parámetros del clasificador obteniendo mejoras considerables en la detección. Otro avance presentado es el algoritmo de *Gentle Boost*, con el que se ha obtenido una mayor tasa de detección evaluando una menor cantidad de *features* [30]. También se han introducido los CART (*Classification And Regression Trees*), que son pequeños árboles de decisión utilizados como clasificadores débiles en lugar de los árboles de un solo nodo o *stumps*. Se ha analizado también una nueva estructura de clasificadores fuertes, en forma de árbol en vez de cascada. La estructura de árbol permite tratar mejor con la inter-variabilidad que se presenta dentro de una clase de objetos como la cara (caras frontales, caras de perfil o inclinadas, caras ocluidas por anteojos, caras ocluidas por bufandas, etc). Finalmente, se presenta una mejora realizada sobre el algoritmo de detección, al momento de escalar el detector para evaluarlo en la imagen. Al escalar un *feature* puede cambiar la relación existente entre las áreas que lo componen y la mejora compensa estos cambios haciendo que se mantenga la relación original que había durante el entrenamiento.

En el capítulo 4, incluimos los resultados obtenidos al realizar diversas pruebas de entrenamiento del detector, empleando distintos parámetros y viendo de qué manera afectan al detector final.

Capítulo 3

Reconocimiento

En los últimos años el reconocimiento de rostros ha sido un tema de gran interés para los investigadores de las áreas de biometría, reconocimiento de patrones y las comunidades de visión por computadora. Este interés en parte se debe a la gran cantidad de aplicaciones que requieren de esta tecnología, entre las que podemos mencionar: vigilancia automática de grupos de personas, seguridad en aeropuertos, páginas web de redes sociales, software embebido en cámaras fotográficas, software de procesamiento de imágenes y administración de fotografías, análisis de videos, diseño de interfaces de computadoras (Human Computer Interface – HCI), sistemas de seguridad y confort en vehículos y hogares, entre otros.

En la actualidad existen diversos métodos de reconocimiento de rostros. Entre los más populares se encuentran *eigenfaces*[44] y *active appearance model*[14][43][26]. En este trabajo abordaremos el reconocimiento de caras mediante un novedoso método que tiene su fundamento en recientes avances en el estudio del procesamiento estadístico de señales, mas específicamente en el área de *compressed sensing*.

Compressed sensing[10] es una teoría alternativa a la ley de Nyquist que busca reconstruir una señal utilizando una menor cantidad de muestras que lo que habitualmente se considera necesario. Para ello aprovecha la existencia de cierta redundancia en la mayoría de las señales de interés. Esta teoría sostiene que una señal dada puede ser representada mediante una combinación lineal esparsa de elementos base (también llamados átomos y/o muestras) almacenados en un diccionario (ver definición 3.3).

Definición 3.1. *Decimos que una descomposición lineal es* ***esparsa*** *cuando la cantidad de átomos utilizados en la combinación lineal es pequeña respecto al total de átomos del diccionario. En caso contrario decimos que la solución es* ***densa****.*

En el contexto del reconocimiento de rostros, disponemos de un dic-

cionario cuyos átomos representan caras de individuos conocidos, y de una imagen de un sujeto a identificar, que llamamos imagen de test. Cada individuo representa una clase. Para efectuar el reconocimiento, buscamos la combinación lineal más esparsa de átomos del diccionario que represente a la imagen de test. Si en dicha combinación lineal la mayor cantidad de átomos utilizados pertenecen a un mismo individuo, podemos decir, con un cierto grado de certeza, que hemos identificado al sujeto de la imagen de test.

A lo largo de este capítulo desarrollaremos el método empleado para reconocer rostros. Para ello, en la sección *3.1 Reconocimiento como una descomposición lineal esparsa*, comenzamos analizando el problema y planteamos formalmente el sistema de ecuaciones que debemos resolver. Posteriormente, en la sección *3.2 Clasificación*, mostramos de qué manera obtenemos la identidad del individuo a testear en función de la solución encontrada. Además, definimos un indicador que nos permitirá saber cuán confiable es el reconocimiento.

Sin embargo, en diversas circunstancias la imagen puede estar parcialmente ocluida, dificultando el reconocimiento. Para evitar que el algoritmo falle, en la sección *3.3 Reconocimiento en Presencia de Oclusión*, abordaremos el problema de reconocer rostros aún si estos estuviesen parcialmente ocluidos. Primero presentamos una modificación al método planteado anteriormente a fin de hacerlo robusto a la oclusión y analizamos algunos inconvenientes del mismo. Debido a esto último, buscamos como alternativa a dicho planteo, detectar la parte ocluida de la imagen a fin de no utilizarla en el reconocimiento.

Luego cerraremos el capítulo con una breve reseña de todo lo presentado en el presente capítulo.

3.1. Reconocimiento como una descomposición lineal esparsa

En esta sección modelamos el problema de reconocimiento de rostros como un problema de optimización en el cual se busca representar la imagen de test como una combinación lineal esparsa de las imágenes del diccionario de caras conocidas.

Todas las imágenes utilizadas tienen el mismo tamaño $ancho \times alto$ y se encuentran normalizadas.

A partir de esta sección, redefinimos imagen de la siguiente manera:

Definición 3.2. *Una **imagen** es un punto en el espacio $\mathbb{R}^m$, donde $m = ancho \times alto$, que se obtiene al apilar sus vectores columnas.*

Se ha demostrado que las imágenes de un mismo rostro bajo diferentes condiciones de iluminación y expresiones caen (aproximadamente) en un subespacio lineal de $\mathbb{R}^m$ de mucha menor dimensión, llamado *subespacio de caras* [3][2][28].

Definición 3.3. *Un **diccionario** de n átomos es una matriz $A \in \mathbb{R}^{m \times n}$ donde cada una de las n columnas es una imagen $\in \mathbb{R}^m$ de un sujeto conocido.*

Sean $A \in \mathbb{R}^{m \times n}$ un diccionario con n átomos, $y \in \mathbb{R}^m$ una imagen de test, buscamos representar la imagen y por medio de una combinación lineal de los átomos de A. Llamamos $x \in \mathbb{R}^n$ a los coeficientes utilizados de la combinación lineal.

$$y = Ax \tag{3.1}$$

Si $m < n$, el sistema de ecuaciones es indeterminado y existe más de un x que satisface la ecuación (3.1)[1]. En la solución tradicional, de todos los posibles x se elige aquel que tiene menor norma l^2 ($||\cdot||_2$), y el problema a resolver queda:

$$\hat{x}_2 = \text{mín}\, ||x||_2 \qquad \text{sujeto a} \quad y = Ax \tag{3.2}$$

Sin embargo, la solución encontrada $\hat{x}_2$ generalmente es densa [15][12], es decir, que gran parte de sus términos no son nulos y, en consecuencia, la combinación lineal $y = Ax$ utiliza gran cantidad de átomos del diccionario pertenecientes a distintas clases. Si bien este sistema nos da una solución, no hace distinción de clases y no nos brinda información para el reconocimiento.

Nuestro objetivo es buscar la solución más esparsa, es decir, aquella que posee la mayor cantidad de términos nulos y utiliza la menor cantidad de átomos del diccionario. Para ello podríamos utilizar la norma l^0 ($||\cdot||_0$). Se define la norma[2] l^0 como:

$$||x||_0 \triangleq \lim_{p \to 0} ||x||_p^p \tag{3.3}$$

donde $||x||_p$ es la norma-p de x y se define como:

[1] Para que la solución sea única, deben agregarse más restricciones.

[2] A pesar de llamarse norma, $||x||_0$ no es verdaderamente una norma, ya que $||\alpha x||_0 = ||x||_0 \neq \alpha ||x||_0$ para $\alpha \neq 0$.

$$||x||_p \triangleq \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \tag{3.4}$$

Nótese que cuando p es igual a 2, se obtiene la fórmula de la norma l^2. Luego, podemos escribir l_0 como:

$$||x||_0 = \lim_{p \to 0} \left(\sum_{i=1}^{n} |x_i|^p \right) \tag{3.5}$$

Si definimos $0^0 \triangleq 0$, entonces la norma l_0 es equivalente a la cantidad de términos no nulos de x. Esto nos permite encontrar la solución más esparsa y el sistema a resolver queda:

$$\hat{x}_0 = \text{mín} \, ||x||_0 \qquad \text{sujeto a } y = Ax \tag{3.6}$$

El inconveniente de este cálculo reside en su complejidad computacional. El problema es en general NP-hard e incluso es difícil hallar una solución aproximada [1].

Sin embargo, recientes estudios en representaciones esparsas y en el estudio de muestreo de señales [15], muestran que para muchas matrices A la solución más esparsa del problema (3.6) puede ser hallada utilizando la norma l^1 ($||\cdot||_1$) en lugar de l^0:

$$\hat{x}_1 = \text{mín} \, ||x||_1 \qquad \text{sujeto a } y = Ax \tag{3.7}$$

donde

$$||x||_1 = \sum_{i=1}^{n} |x_i|$$

Este último problema es un problema de optimización lineal y se puede considerar tratable en un tiempo razonable, ya que puede resolverse en tiempo polinomial. Se puede modelar de la siguiente manera:

$$\text{mín} \sum_{i}^{n} |x_i|$$

$$\text{sujeto a} \quad Ax = y$$

que, haciendo un cambio de variable, es lo mismo que:

$$\text{mín} \sum_{i}^{n} s_i$$

$$\text{sujeto a} \quad \begin{cases} |x_i| & = & s_i & \text{con} & i = 1 \dots n \\ Ax & = & y \end{cases}$$

Dado que estamos minimizando $\sum_i s_i$ la formulación anterior puede reescribirse de la siguiente manera:

$$\text{mín} \sum_{i}^{n} s_i$$

$$\text{sujeto a} \quad \begin{cases} |x_i| & \leq & s_i & \text{con} & i = 1 \dots n \\ Ax & = & y \end{cases}$$

Teniendo en cuenta que $|x_i| \leq s_i \Leftrightarrow -s_i \leq x_i \leq s_i$, queda:

$$\text{mín} \sum_{i}^{n} s_i$$

$$\text{sujeto a} \quad \begin{cases} x_i & \leq & s_i & \text{con} & i = 1 \dots n \\ -x_i & \leq & s_i & \text{con} & i = 1 \dots n \\ Ax & = & y \end{cases}$$

El sistema anterior puede optimizarse reemplazando s_i por $t_i = s_i - x_i$. Finalmente, obtenemos:

$$\text{mín} \sum_{i} x_i + t_i$$

$$\text{sujeto a} \quad \begin{cases} t_i & \geq & 0 & \text{con} & i = 1 \dots n \\ 2x_i + t_i & \geq & 0 & \text{con} & i = 1 \dots n \\ Ax & = & y \end{cases} \tag{3.8}$$

Esta última formulación puede modelarse con suma facilidad utilizando alguna librería de programación lineal[3]. Una vez resuelto el sistema y obtenido el valor de x podremos identificar la identidad del individuo utilizando un método que describiremos más adelante en la sección 3.2. El inconveniente que presenta esta formulación es que en presencia de ruido puede no ser lo suficientemente estable, haciendo que la solución del mismo deje de ser esparsa. Por ello, en la siguiente sección replantearemos el problema consiguiendo un sistema más estable que contempla el ruido.

[3]En el presente trabajo decidimos utilizar la librería de optimización Mosek. La misma puede conseguirse en http://www.mosek.com

3.1.1. Modelando el Ruido Gaussiano

En la ecuación (3.1) asumimos que la imagen podía expresarse en forma exacta utilizando los átomos del diccionario A y en (3.7) buscamos que dicha descomposición sea esparsa. Sin embargo, las imágenes reales contienen ruido y eso puede producir que la imagen y no pueda ser expresada en forma exacta mediante unos pocos átomos del diccionario. Podemos entonces reformular la ecuación (3.1) de la siguiente manera:

$$y = Ax + z \ , \tag{3.9}$$

donde $z \in \mathbb{R}^m$ representa el ruido cuya energía está acotada por $||z||_2 \leq \epsilon$. Reformulemos entonces la ecuación (3.7):

$$\hat{x}_1 = \text{mín}\, ||x||_1 \ , \qquad \text{sujeto a} \quad ||Ax - y||_2 \leq \epsilon \ . \tag{3.10}$$

Si replanteamos el problema de programación lineal (3.8) para incluir el ruido, nos queda el siguiente sistema:

$$\begin{aligned} &\text{mín} \sum_i x_i + t_i \\ &\text{sujeto a} \begin{cases} t_i & \geq & 0 & \text{con } i = 1 \dots n \\ 2x_i + t_i & \geq & 0 & \text{con } i = 1 \dots n \\ ||Ax - y||_2 & \leq & \epsilon \ . \end{cases} \end{aligned}$$

Introducimos una variable $b = Ax - y$ a fin de simplificar el sistema.

$$\begin{aligned} &\text{mín} \sum_i x_i + t_i \\ &\text{sujeto a} \begin{cases} t_i & \geq & 0 & \text{con } i = 1 \dots n \\ 2x_i + t_i & \geq & 0 & \text{con } i = 1 \dots n \\ Ax - y & = & b \\ ||b||_2 & \leq & \epsilon \end{cases} \end{aligned}$$

La mayoría de las librerías que permiten modelar este tipo de sistemas prohíben utilizar constantes en la última ecuación. Es por ello que introducimos una variable nueva c y agregamos una restricción $c = \epsilon$. Si además de esto, desarrollamos la norma l^2 obtenemos la siguiente formulación:

$$\text{mín} \sum_i x_i + t_i$$

$$\text{sujeto a} \quad \begin{cases} t_i & \geq & 0 & \text{con } i = 1 \dots n \\ 2x_i + t_i & \geq & 0 & \text{con } i = 1 \dots n \\ Ax - y & = & b & \\ c & = & \epsilon & \\ \sqrt{\sum_{i=1}^{m} b_i^2} & \leq & c\ , & \end{cases} \tag{3.11}$$

donde podríamos pensar la última ecuación como la cónica $\mathcal{C}$ definida de la siguiente manera:

$$\mathcal{C} \triangleq \left\{ b \in \mathbb{R}^m,\ c \in \mathbb{R} : c \geq \sqrt{\sum_{i=1}^{m} b_i^2} \right\}$$

Esta función cónica es conocida como el Cono de Lorenz o también como "cono de segundo orden" ("Second Order Cone" en inglés). Como se puede apreciar, a diferencia de la formulación (3.8), la formulación (3.11) no es lineal. Este tipo de problemas son conocidos como Second Order Cone Programming (SOCP) [13], los cuales pueden ser resueltos en forma eficiente a pesar de no ser lineales.

Esta última formulación es más robusta que el sistema 3.8 cuando las imágenes poseen ruido gaussiano.

3.1.2. Selección de *features*

En la sección 3.1 planteamos que la ecuación 3.1 tiene múltiples soluciones si $m < n$, donde $A \in \mathbb{R}^{m \times n}$. Ahora bien, considerando que habitualmente se tienen unas pocas imágenes por individuo, si decidimos emplear imágenes de 640×480 probablemente nuestro diccionario A tenga $m \gg n$ (ya que m resulta del orden de 10^5).

Cuando $m > n$, el sistema está sobredeterminado y puede que no tenga solución. Habitualmente en estos casos se busca la solución de *mínimos cuadrados* minimizando $||y - Ax||_2$, pero dicha solución no es esparsa. Para solucionar este problema, buscamos reducir la dimensión de las imágenes de m a $d \ll n$ y luego calcular la representación esparsa de la imagen de test y en un espacio de menor dimensión.

Existen numerosos métodos en la literatura de visión por computadora que

fueron investigados para proyectar imágenes de gran dimensión en un espacio de *features* (características) pequeño. Algunos de ellos tratan de obtener características generales de la cara, como por ejemplo Eigenfaces [44], Fisherfaces [3] y Laplacianfaces [24]. Otros métodos tratan de extraer características faciales locales (por ejemplo, el borde de los ojos) [41], [38]. Todas estas proyecciones desde el espacio de las imágenes al espacio de características pueden representarse como una matriz $R \in \mathbb{R}^{d \times m}$ con $d \ll m$.

Aplicando R a ambos lados de la ecuación 3.1 nos queda:

$$\tilde{y} \triangleq Ry = RAx \quad \in \mathbb{R}^d \tag{3.12}$$

Si definimos $\tilde{A} \triangleq RA \ \in \mathbb{R}^{d \times n}$ obtenemos

$$\tilde{y} = \tilde{A}x \quad \in \mathbb{R}^d \tag{3.13}$$

Este nuevo sistema es indeterminado, ya que definimos $d \ll n$, y en consecuencia tiene más de una solución. Es decir, que estamos ante un sistema de ecuaciones con las mismas características que los descriptos en las secciones 3.1 y 3.1.1 y en consecuencia su desarrollo es análogo.

A pesar de haber numerosos trabajos que proponen diferentes métodos para extraer *features*, existe muy poco consenso en la comunidad científica acerca de cuáles son mejores y cuáles peores. Lamentablemente tampoco hay fundamentos que nos ayuden a decidir que método emplear en cada caso [49]. Sin embargo, la teoría de *compressed sensing* nos brinda una nueva perspectiva acerca de la selección de *features*: esta sugiere que la cantidad de *features* es más importante que la forma en que estos son obtenidos. Es decir, que mientras que la cantidad de *features* sea lo suficientemente grande, aún *features* elegidos en forma aleatoria son suficientes para recuperar la representación esparsa [15], [9].

Debido a esto, en [49] se comparan cinco métodos para extraer *features*: Eigenfaces, Laplacianfaces, Fisherfaces, Downsample y Random Faces (ver sección 3.1.2.1). Estos métodos fueron comparados con cuatro dimensiones d diferentes. Los resultados confirmaron el marco teórico de *compressed sensing* ya que la precisión aumentó significativamente a medida que d aumentaba. Adicionalmente, los resultados mostraron que hay muy poca diferencia en la tasa de reconocimiento entre los diversos métodos para un tamaño de d dado.

Debido a que prácticamente no hay diferencia en la tasa de reconocimiento entre los métodos, nosotros elegimos Random Faces ya que tiene algunas ventajas sobre el resto.

3.1.2.1. Random Faces

Random Faces consiste en tomar features al azar de las imágenes. Consideremos $R \in \mathbb{R}^{d \times m}$ donde los elementos de la matriz son números aleatorios independientes entre sí con distribución normal de media cero. Luego, cada fila de la matriz es normalizada a una unidad. La matriz R puede verse como d random faces en R^m.

La idea de tener proyecciones aleatorias no es algo nuevo, sino que por el contrario existen numerosos trabajos que han utilizado exitosamente esta técnica [27], [4].

Como mencionamos anteriormente, este método tiene varias ventajas respecto de los otros estudiados en [49].

1. A diferencia de otros métodos, no requiere de un conjunto de entrenamiento de caras para generar la matriz R. Es más, esta matriz es independiente de las caras del diccionario.

2. La matriz R puede generarse eficientemente aún cuando d es grande.

Una gran ventaja de que R sea independiente de las caras presentes en el diccionario es que permite agregar nuevas caras a la misma sin la necesidad de recalcular dicha matriz. Esto permite que el diccionario de caras pueda crecer en forma incremental a un bajo costo computacional.

Lo único que hay que tener en cuenta al utilizar Random Faces es la estabilidad. Puede pasar que para un individuo dado, los features seleccionados no sean apropiados [18]. Si bien los resultados obtenidos en [49] muestran que esto rara vez ocurre, se puede conseguir una mayor estabilidad utilizando múltiples Random Projections y luego consolidando estos resultados.

En la siguiente sección vemos cómo obtener la identidad del individuo en función de los resultados obtenidos a partir de la formulación (3.11).

3.2. Clasificación

Una vez resuelto el sistema (3.11), debemos determinar si el sujeto testeado se encuentra en el diccionario y, en tal caso, determinar la identidad del mismo en función del resultado obtenido. Antes mencionamos que, en el caso ideal y asumiendo que el sujeto se encuentra en nuestro diccionario de caras, la solución que se obtiene está compuesta exclusivamente por átomos de la clase correspondiente a dicho sujeto. Sin embargo, esto rara vez ocurre por

diversas razones: cambios significativos en la expresión respecto a las muestras de entrenamiento, ruido en la imagen y/o oclusión.

Se ha observado que, cuando el sujeto a evaluar no se encuentra representado en el diccionario, la solución obtenida presenta coeficientes esparsos distribuidos a lo largo de los átomos de los diferentes sujetos presentes en el diccionario. En cambio, si el individuo a evaluar se encuentra en el diccionario, la solución presenta coeficientes concentrados en sus correspondientes átomos.

Esta información resulta útil para, en primer lugar, determinar si la persona a identificar se encuentra en el diccionario, y en segundo lugar, para determinar su identidad. Es por ello que buscamos una forma para medir cuán concentrados están los coeficientes en un individuo dado.

Sean $W_1, W_2, \ldots, W_k$, con $k > 1$, las distintas clases o individuos presentes en el diccionario A. Se define entonces, el SCI (Sparsity Concentration Index) de un vector $x \in \mathbb{R}^n$ como:

$$SCI(x) \triangleq \frac{k \cdot \max_j \left\|\delta_{W_j}(x)\right\|_1 / \|x\|_1 - 1}{k-1} \quad \forall j = 1 \ldots k \tag{3.14}$$

donde

$$\delta_{W_j}(x) : \mathbb{R}^n \to \mathbb{R}^n \wedge x = (x_1, x_2, \ldots, x_n)^t \in \mathbb{R}^n$$

Llamemos $A(i)$ a la imagen del diccionario A que se corresponde con el coeficiente x_i. Luego, definamos $\delta_{W_j}(x)$ de la siguiente manera:

$$\begin{aligned} \delta_{W_j}(x) &= \left(x_1^j, x_2^j, \ldots, x_n^j\right)^t \\ x_i^j &= \begin{cases} x_i & \text{Si } A(i) \in W_j \\ 0 & \text{En otro caso} \end{cases} \qquad \forall i = 1 \ldots n \end{aligned} \tag{3.15}$$

Es decir, $\delta_{W_j} : \mathbb{R}^n \to \mathbb{R}^n$ es la función característica que, dado un vector $x \in \mathbb{R}^n$, devuelve otro vector $x^j \in \mathbb{R}^n$ igual a x pero haciendo nulos aquellos términos que no se corresponden con los átomos de la clase W_j.

Si la imagen está representada utilizando átomos de una única clase, nos queda $SCI(x) = 1$. Por el contrario, si los coeficientes están dispersos uniformemente en las distintas clases, obtenemos $SCI(x) = 0$.

Para determinar si el sujeto evaluado se encuentra en el diccionario, se establece un umbral τ sobre el SCI, de tal manera que si $SCI(x) \geq \tau$, entonces

se considera al sujeto como conocido (dentro del diccionario).

Una vez que se determina que el sujeto se encuentra en el diccionario, se procede a identificar el mismo. Para ello definamos el residuo r_{W_j} de la siguiente manera:

$$r_{W_j}(y) \triangleq \left\| y - A\,\delta_{W_j}(x) \right\|_2 \tag{3.16}$$

El residuo r_{W_j} nos permite tener una idea de cuanto se asemeja la imagen de test y con respecto a la imagen generada por los átomos correspondientes a la clase W_j.

Utilizando el residuo r_{W_j} podemos definir la identidad del individuo como la clase W_j que minimiza el residuo:

$$Identidad(y) \triangleq \arg\min_{W_j} \; r_{W_j}(y) \tag{3.17}$$

En la sección 3.1.2.1 vimos que podemos mejorar la estabilidad del algoritmo utilizando múltiples Random Faces. Es decir, que tenemos $R^1, R^2, \ldots, R^L$ random faces. En consecuencia tendremos múltiples residuos, uno por cada R^l, osea $r^l_{W_j}$ con $l = 1, \ldots, L$. Redefinimos entonces la identidad del individuo como la clase W_j que:

$$Identidad(y) \triangleq \arg\min_{W_j} \left(\frac{1}{L} \sum_{l=1}^{L} r^l_{W_j}(y) \right)$$

De esta manera, utilizando los indicadores $SCI\,(x)$ y $Identidad\,(y)$ podemos saber si el sujeto de la imagen y se encuentra en el diccionario y en el caso que lo esté, saber quién de ellos es.

3.3. Reconocimiento en Presencia de Oclusión

En la sección 3.1 modelamos el problema de reconocimiento de rostros como un problema de optimización lineal. Luego en la sección 3.1.1 modificamos dicho planteo a fin de hacerlo más robusto al ruido gaussiano. Sin embargo, ninguna de estas formulaciones es capaz de reconocer individuos cuando estos tengan una cierta porción de la imagen ocluida (por ejemplo, con anteojos de sol).

La oclusión por bloques suele presentarse comúnmente en las imágenes, introduciendo errores de gran magnitud. En consecuencia, estos errores no

pueden ser modelados mediante la formulación $y = Ax + z$ (3.9), diseñada más bien para la presencia de ruido pequeño.

En consecuencia, se puede pensar a la oclusión como un error e que afecta a una porción de tamaño ρ de la imagen. De esta manera, se reformula (3.1) de la siguiente manera:

$$y = Ax + e \tag{3.18}$$

donde e solo tiene componentes no nulas en la porción ocluida de la imagen. La ubicación de dichos errores no es conocida y su magnitud es completamente arbitraria. Sin embargo, asumimos que la porción afectada por la oclusión es pequeña en relación a la imagen completa.
Se puede reescribir (3.18) como:

$$y = \begin{bmatrix} A, I \end{bmatrix} \begin{bmatrix} x \\ e \end{bmatrix} = Bu \tag{3.19}$$

donde

$$B = \begin{bmatrix} A, I \end{bmatrix} \in \mathbb{R}^{m \times (n+m)}$$

$$u = \begin{bmatrix} x \\ e \end{bmatrix}$$

El sistema (3.19) es indeterminado y, si existe alguna solución para u, dicha solución no es única. La solución más esparsa $u_0 = \begin{bmatrix} x_0, e_0 \end{bmatrix}$, obtenida utilizando la norma l^0, está compuesta por a lo sumo $n_i + \rho m$ términos no nulos, siendo n_i la cantidad de átomos del diccionario asociados a la persona a reconocer. Esta solución es lo suficientemente esparsa como para utilizar la norma l^1 y obtener la misma solución. Por lo tanto, se reformula el sistema (3.7), extendiéndolo a:

$$\hat{u}_1 = \text{mín} \, ||u||_1 \qquad \text{sujeto a } Bu = y \tag{3.20}$$

Es interesante mencionar que al modelar el problema de esta manera el sistema siempre queda indeterminado (como $n > 1$ resulta que $m < n + m$), lo cual no ocurría en la sección 3.1.2. Esto nos permite poder trabajar con toda la imagen y de esta manera aprovechar al máximo la redundancia de información presente en la parte no ocluida de la cara.

Luego, a fin de poder calcular la identidad del individuo detectado, necesitamos reformular también la ecuación del residuo (3.16). La nueva ecuación queda:

$$r_{W_j}(y) \triangleq \left\| y - \hat{e}_1 - A\,\delta_{W_j}(\hat{x}_1) \right\|_2 \tag{3.21}$$

donde

$$\hat{u}_1 = [\hat{x}_1, \hat{e}_1]$$

Este modelo funciona muy bien cuando la oclusión se encuentra dispersa por toda la imagen, como ser una gran cantidad de ruido impulsivo. Sin embargo, cuando la oclusión se encuentra concentrada en un lugar, el método comienza a fallar si la imagen está ocluida más del 33 % [48]. En la siguiente sección veremos como mejorar el reconocimiento cuando se excede dicho porcentaje de oclusión.

3.3.1. Particionar el problema

La oclusión generalmente suele presentarse en bloques, es decir que se encuentra concentrada en una o más regiones de la imagen. Esto se debe a que el motivo principal de la oclusión suele ser que cierta porción de la cara no puede verse debido a que hay algo que la tapa (ya sean anteojos, pelo, bufandas, etc...). Por este motivo nos es de especial interés que el reconocimiento sea robusto a este tipo de oclusión.

El método que vimos en la sección anterior se vuelve más inestable e impreciso a medida que aumenta el porcentaje ocluido en las imágenes de test. Para mejorar este inconveniente, [48] propone subdividir el problema en problemas más pequeños. Para ello se particiona la imagen en bloques de menor tamaño y luego se procesa cada bloque por separado. Una vez obtenidas las identidades reconocidas por cada bloque, se utiliza un sistema de votación para finalmente determinar la identidad de la persona. De esta manera, se espera que al presentarse oclusión en ciertas zonas del rostro, la misma afecte solo a unas pocas particiones de la imagen y el resto de las particiones puedan determinar correctamente la identidad. Además se puede detectar qué particiones están más afectadas por la oclusión a través del SCI medido en cada una, y descartar esos votos. En cada una de las particiones se calcula el sistema (3.20), ya que puede ocurrir que una zona de la partición se encuentre ocluida.

Específicamente, se particiona la imagen a evaluar y y cada imagen del diccionario de caras en L bloques de tamaño $a \times b$. A partir del diccionario particionado se obtienen las matrices: $A^{(1)}, A^{(2)}, \ldots, A^{(L)} \in \mathbb{R}^{p \times n}$, con $p = ab$, y se plantea para cada bloque el sistema: $y^{(l)} = A^{(l)}x^{(l)} + e^{(l)}$, con $y^{(l)}, x^{(l)}, e^{(l)} \in \mathbb{R}^p$. De la misma manera, calculamos para cada bloque el sistema de la

ecuación 3.20, obteniendo:

$$\hat{u}_1^{(l)} = \text{mín}\,||u||_1 \qquad \text{sujeto a } B^{(l)}u = y^{(l)} \tag{3.22}$$

donde

$$u \in \mathbb{R}^{n+p} \quad \wedge \quad u = \begin{bmatrix} x^{(l)} \\ e^{(l)} \end{bmatrix}$$

$$B^{(l)} = \begin{bmatrix} A^{(l)} I \end{bmatrix}$$

Dividir la imagen en porciones más pequeñas puede ayudar a reconocer satisfactoriamente cuando un gran porcentaje de la cara se encuentra ocluido. Sin embargo, esto también trae aparejados algunos problemas.

El principal problema que esto presenta es que no se puede decidir cuál es la mejor partición que se debe efectuar para que la oclusión no afecte a gran parte de los bloques [50]. Si bien cada bloque es resistente a cierto porcentaje de oclusión, gracias al método utilizado que la contempla, muchas veces se superará ese límite y más aún tratándose de porciones más chicas de la imagen. En los casos en que la partición posea gran parte ocluida, se descartará dicha partición perdiendo la información presente en la pequeña fracción no ocluida.

El segundo inconveniente de este método es que emplea para el reconocimiento únicamente las características locales presentes en cada una de las partes, dejando de lado la información global existente entre ellas. Esto hace que el reconocimiento en cada una de las partes sea menos estable. Para dar un ejemplo, supongamos que los ojos de un sujeto dado estén cubiertos por particiones distintas. Es de esperar, que al hallar la solución utilizando el sistema sin particionar, los ojos recuperados en el reconocimiento provengan de la misma imagen y, en consecuencia, del mismo sujeto, mientras que al utilizar el sistema particionado provengan de imágenes distintas con ojos similares y posiblemente de distintos sujetos.

Debido a estos problemas, en la siguiente sección presentamos una forma alternativa para reconocer imágenes con mucha oclusión.

3.3.2. Detección de la Oclusión

En esta sección buscamos enfocar el reconocimiento de caras con oclusión de una manera distinta. En las secciones 3.3 y 3.3.1 tratábamos de desarrollar un método que sea lo suficientemente robusto para reconocer la identidad

de la persona aún cuando la imagen tuviese oclusión. Nuestro método por el contrario, busca detectar qué partes de la cara se encuentran ocluidas para luego excluirlas en el reconocimiento. Haciendo esto, podremos utilizar toda la información no ocluida de la imagen para reconocer al individuo.

En la actualidad existen otros métodos que buscan detectar la oclusión. Por ejemplo en [32] se utiliza un método de filtros bayesianos y de *GraphCut* para luego hacer *Face Inpainting* (restaurar las secciones ocluidas). Además, en [50] se presenta otro método que detecta la oclusión y reconoce al individuo utilizando *Markov Random Fields*, pero con un costo computacional elevado. En este trabajo empleamos para detectar la oclusión los mismos conceptos de la teoría de *compressed sensing* que venimos utilizando.

El problema de detectar la oclusión consiste en determinar qué píxeles de una imagen muestran una porción de un rostro y cuales no. Podemos pensar a los píxeles ocluidos como valores atípicos en un rostro. Detectar la oclusión entonces, es detectar aquellos píxeles con valores atípicos para el rostro de la imagen.

Consideremos una imagen con oclusión sintética, como la que se muestra en la figura 3.1(a). Supongamos que de alguna forma, conseguimos la misma imagen pero sin oclusión (b). Entonces, haciendo la diferencia entre ambas (y tomando el valor absoluto) obtendremos una imagen donde los píxeles no nulos son aquellos que fueron afectados por la oclusión (c). Si umbralizamos dicha imagen tomando un umbral $\tau = 1$, obtendremos la imagen (d) que muestra todos los píxeles de la imagen (c) mayores a cero, los cuales en este caso demarcan perfectamente el área ocluida.

(a) (b) (c) (d)

Figura 3.1: **(a)** Imagen ocluida en forma sintética. **(b)** Imagen sin ocluir. **(c)** Resultado de la diferencia entre (a) y (b). **(d)** Resultado de umbralizar con $\tau = 1$ la imagen (c).

De más está decir que la situación planteada en la imagen 3.1 es irreal ya que en la práctica resulta imposible tener la misma imagen con y sin oclusión. Analicemos entonces qué sucede si en vez de tener la misma imagen pero sin oclusión, tenemos otra imagen del mismo sujeto, con la misma pose, expresión e iluminación.

En la figura 3.2(c) podemos ver un ejemplo de lo que sucede cuando

(a) (b) (c) (d)

Figura 3.2: **(a)** Imagen ocluida en forma sintética. **(b)** Otra imagen del mismo sujeto, con pose, expresión e iluminación similares. **(c)** Resultado de la diferencia entre (a) y (b). **(d)** Resultado de umbralizar con $\tau = 50$ la imagen (c).

utilizamos otra imagen del mismo sujeto. Las zonas que no se encuentran ocluidas ya no tienen 0 en sus píxeles, sin embargo las zonas ocluidas tienen una mayor intensidad en la imagen. Por este motivo, tenemos que elegir un umbral τ mayor. La imagen (d) fue obtenida utilizando un umbral $\tau = 50$.

La imagen 3.2(d) presenta varios aspectos interesantes a destacar. El primero es que aparecen pequeñas secciones erróneamente detectadas como oclusión en el área de la boca y la nariz. Esto puede reducirse si en vez de comparar píxel a píxel, hubiésemos comparado vecindades de píxeles. En segundo lugar, podemos ver que en la esquina superior derecha de la imagen aparece marcado como oclusión un poco de pelo. El pelo que cubre esa porción de la frente no aparece en la imagen 3.2(a) sino en (b). Es decir, que la oclusión detectada incluye las partes cubiertas de ambas imágenes. En tercer y último lugar, podemos ver que la zona en la que se encuentran los iris de los ojos no está marcada como oclusión. Esto se debe a que el color negro de la oclusión sintética de 3.2(a) no es un valor atípico para el iris de esta persona. La figura 3.3 muestra un ejemplo de esto.

Al ver la imagen ocluida en forma sintética 3.3(g) lo primero que viene a nuestra mente es que la oclusión es un rectángulo negro en la zona de los ojos. Sin embargo, como veíamos antes, el algoritmo no marca como zonas ocluidas el iris del individuo. Para responder la pregunta si ese resultado tiene sentido, realizamos la siguiente prueba. Primero obtenemos la sección correspondiente a los ojos detectada como oclusión, multiplicando la misma 3.3(a) con la oclusión sintética rectangular (b) y obtenemos (c). Luego, utilizando esa máscara, pintamos la imagen sin ocluir (e) con negro. Si comparamos el resultado obtenido (f) con la imagen ocluida original (g), observamos que prácticamente no hay diferencias. Podemos concluir entonces que la máscara detectada para la zona de los ojos es correcta, ya que al ocluir la sección detectada obtenemos un resultado muy similar al original.

Ahora bien, en los ejemplos mencionados anteriormente utilizábamos una im-

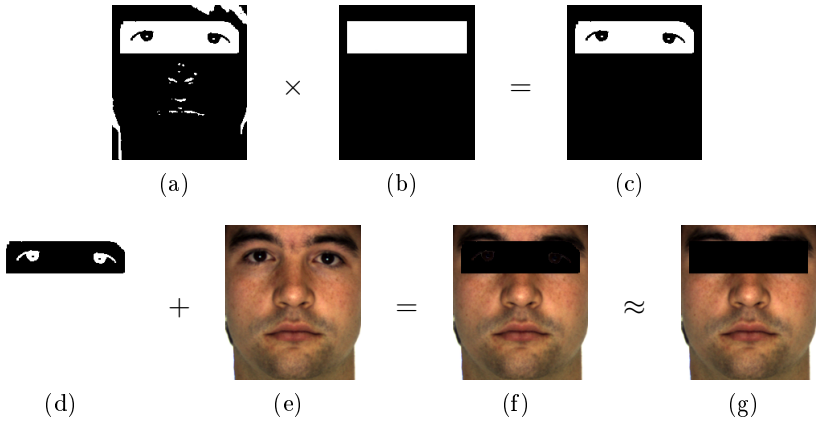


Figura 3.3: **(a)** Máscara obtenida en 3.2(d). **(b)** Máscara de la oclusión sintética agregada a la imagen (g). **(c)** Resultado de multiplicar (b) y (a), a fin de quedarnos sólo con la oclusión detectada en la zona de los ojos. **(d)** Imagen (c) invertida con fondo transparente. **(e)** Imagen original sin ocluir. **(f)** Resultado de ocluir la imagen (e) con (d) . **(g)** Imagen original ocluida sintéticamente. Puede observarse que no es muy distinta de la imagen (f).

agen del mismo sujeto y con condiciones similares (expresión, iluminación, etc...) a la imagen ocluida. Esto puede parecer contradictorio si se tiene en cuenta que la identidad del individuo es lo que buscamos averiguar. Por este motivo, en vez de contar con dicha imagen, buscamos generarla a partir de las imágenes presentes en el diccionario.

Supongamos que de alguna manera logramos identificar un fragmento (ver definición 3.4) de la imagen que no se encuentra ocluido, o en su defecto que tiene una proporción pequeña de píxeles ocluidos.

Definición 3.4. *Un **fragmento** I_F de una imagen I es otra imagen compuesta por un subconjunto F de los píxeles de I.*

Utilizando dicho fragmento, podemos intentar reconocer la identidad de la persona mediante el algoritmo descripto en la sección 3.3. Al utilizar sólo ese fragmento de la imagen, sin duda obtendremos una menor precisión ya que no se utilizarán todos los píxeles no ocluidos dentro del algoritmo. Es más, la precisión dependerá entre otras cosas, de la cantidad de información contenida en dicho fragmento, lo cual guarda cierta relación con el tamaño del mismo. Ahora bien, aparte de obtener una posible identidad del sujeto, obtenemos el vector de coeficientes x que se utilizó para representar dicho fragmento a partir del diccionario. Usando esos coeficientes, podemos obtener una posible imagen similar a la que se busca detectar la oclusión. Veamos esto en detalle en el algoritmo 3.1.

En el paso 4 del algoritmo 3.1, utilizamos los coeficientes x devueltos por el reconocimiento para formar una nueva imagen a partir de la combinación

Algoritmo 3.1 Algoritmo para generar una imagen similar a y utilizando solo los píxeles F

Require: F, un conjunto de píxeles

1. Sea y_F el fragmento F de la imagen de test y.
2. Sea A_F el diccionario que resulta de seleccionar el fragmento F en cada imagen del diccionario A.
3. Utilizando el algoritmo descripto en la sección 3.3, la imagen y_F y el diccionario de caras A_F, calculamos los coeficientes x y el error e tal que $y_F = A_F\ x + e$.
4. **return** $y' = Ax$

lineal de las caras del diccionario. La idea detrás de este paso es que si utilizando x en la combinación lineal $y_F = A_F\ x + e$ logramos representar el fragmento y_F mediante átomos de A_F, podemos extender dicha combinación lineal a toda la imagen y probablemente se asemeje a y.

Anteriormente suponíamos que el fragmento y_F no tenía píxeles ocluidos. Sin embargo, el algoritmo descripto en la sección 3.3, utilizado en el paso 3 del algoritmo 3.1, soporta una proporción no muy grande de oclusión, por lo que el razonamiento sigue siendo válido aún cuando el fragmento y_F tenga algunos píxeles ocluidos.

Si por el contrario, el fragmento y_F tiene una gran proporción de píxeles ocluidos tal que el algoritmo utilizado en el paso 3 comienza a fallar, entonces el error e comenzará a ser grande, por lo que $y_F \not\approx A_F\ x$. En consecuencia es de esperar que la imagen resultante $y' = Ax$ tampoco se aproxime a y.

Buscamos entonces un conjunto de píxeles F que contenga una gran proporción de píxeles no ocluidos para poder utilizar el algoritmo 3.1 y obtener así una imagen y' similar a la imagen de test y, para luego obtener la oclusión presente en esta última. Además, buscamos que F tenga una cantidad significativa de píxeles a fin de que el paso 3 del algoritmo tenga la suficiente información como para devolver un resultado medianamente preciso.

Como no tenemos ninguna información acerca de cómo obtener dicho conjunto de píxeles F, probamos con múltiples conjuntos. Existen diversas formas de obtener estos conjuntos, cada una de ellas tiene sus ventajas y desventajas. En este trabajo consideramos los siguientes métodos:

- Tomar una muestra de píxeles distribuidos uniformemente.
- Tomar una muestra de píxeles aleatorios.
- Tomar bloques de píxeles contiguos en forma aleatoria.

- Utilizar conjuntos de píxeles predefinidos, teniendo en cuenta las oclusiones más comunes que pueden presentarse.

La oclusión suele presentarse en bloques, es decir, que generalmente se encuentra concentrada en una o más porciones de la imagen. Utilizaremos como ejemplo para la explicación la figura 3.4(a) que tiene una oclusión en bloque del 50 %. Esta imagen tiene la oclusión concentrada en un único bloque, pero las conclusiones que obtendremos luego, serán válidas aún cuando la oclusión se encuentre en varios bloques. Para facilitar la lectura, de aquí en adelante marcamos con rojo y azul los píxeles ocluidos y no ocluidos respectivamente, tal como se muestra en la figura 3.4(b).

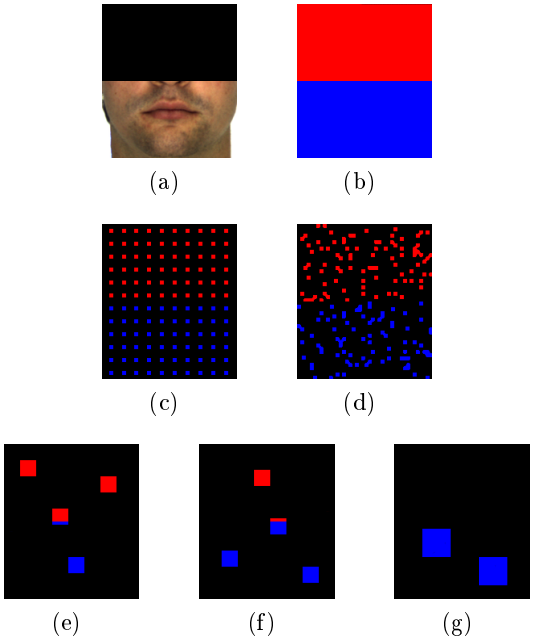


Figura 3.4: Distintas formas de obtener un fragmento I_F. **(a)** Imagen ocluida en un 50 %. **(b)** Los píxeles rojos representan los píxeles ocluidos, mientras que los azules representan a los no ocluidos. **(c)** fragmento resultante al seleccionar píxeles de una manera uniforme. El porcentaje de píxeles ocluidos del fragmento es el mismo que en toda la imagen. **(d)** fragmento resultante al seleccionar píxeles de una manera aleatoria. El porcentaje de píxeles ocluidos del fragmento es similar al de toda la imagen. En este caso es del 52,3 %. **(e)**, **(f)** y **(g)** fragmentos resultantes al seleccionar bloques píxeles en forma aleatoria. El porcentaje de píxeles ocluidos de los distintos fragmentos varía significativamente. Los mismos son: 70 %, 30 % y 0 % respectivamente.

Analicemos entonces el primer método para obtener un conjunto de píxeles F, que consiste en tomar una *muestra uniformemente distribuida* de los píxeles de la imagen. Por ejemplo, en la figura 3.4(c) podemos ver que se eligieron píxeles esparcidos por toda la imagen. Este método tiene la ventaja de que captura muy bien las características globales de la imagen, sin embargo tiene dos grandes desventajas. La primera es que no captura bien

características locales de la imagen. La segunda y principal desventaja es que la proporción de píxeles ocluidos en el conjunto resultante F es la misma (o muy similar) que en la imagen original. Es decir, que si la imagen de test tiene una gran cantidad de píxeles ocluidos (al igual que en el ejemplo que se ve en la figura, donde el 50 % de los píxeles están ocluidos), el conjunto F también los tendrá, lo que producirá que el algoritmo 3.1 tenga un error muy grande y la imagen resultante del mismo difiera considerablemente de la imagen de test en las zonas no ocluidas.

Nuestro segundo método, es un intento un tanto ingenuo de solucionar los problemas del primero y consiste en *seleccionar píxeles en forma aleatoria*. Como puede observarse en la figura 3.4(d), si la cantidad de píxeles a seleccionar es lo suficientemente grande, muy probablemente los píxeles seleccionados se encuentren esparcidos por toda la imagen. Esto hace que este método tenga las mismas ventajas y desventajas que el anterior.

El tercer método consiste en seleccionar *bloques de píxeles contiguos en forma aleatoria*. Cuanto más grandes sean dichos bloques de píxeles, menos bloques serán necesarios para un tamaño de F dado. Entonces, cuanto más bloques y más pequeños sean, el conjunto F tendrá píxeles esparcidos por toda la imagen. Si por el contrario existen pocos bloques de gran tamaño, los píxeles del conjunto F van a estar más concentrados en algunas regiones de la imagen. En las figuras 3.4(e), (f) y (g) se muestran ejemplos de bloques de píxeles seleccionados en forma aleatoria. Si tenemos en cuenta que la oclusión generalmente se encuentra concentrada en regiones de la imagen, eligiendo distintas posiciones de unos pocos bloques de píxeles, obtenemos conjuntos de píxeles con diferentes proporciones de píxeles ocluidos. Por ejemplo, 3.4(e) tiene el 70 % de píxeles ocluidos, 3.4(f) tiene el 30 % y 3.4(g) no tiene ninguno (0 %). Ahora bien, el hecho de que el conjunto de píxeles F se encuentre concentrado en una determinada región puede tener su efecto negativo en el algoritmo 3.1 y hacerlo un poco menos estable. Lamentablemente no hay una forma de determinar la cantidad de bloques óptima, es por ello que resulta conveniente probar con múltiples tamaños y cantidades de bloques.

El cuarto y último método consiste en utilizar *conjuntos de píxeles predefinidos*, teniendo en cuenta las oclusiones más comunes que pueden presentarse. Es decir, que si muchas de las imágenes de test tienen anteojos negros, resulta conveniente probar con un conjunto de píxeles F que no incluya píxeles cercanos a la posición de los ojos [4]. Como es de esperar, este método sirve sólo para ser combinado con otros, ya que no es posible conocer todos

[4] Esto no resulta difícil, ya que asumimos que todas las imágenes se encuentran alineadas.

los tipos de oclusión que pueden presentarse.

Dado que tenemos múltiples conjuntos de píxeles, aplicamos el algoritmo 3.1 varias veces (una por cada conjunto de píxeles). Anteriormente dijimos que si utilizamos en el algoritmo algoritmo 3.1 un conjunto de píxeles que tenga una alta proporción de píxeles ocluidos, la imagen resultante muy probablemente sea muy distinta a la imagen de test. Teniendo en cuenta esto, de todas las imágenes obtenidas $y'_1, \ldots, y'_n$ buscaremos aquella que cumple con:

$$y'_{max} = \arg \max_{y'_i} S_\tau \left(y, y'_i\right) \ ,$$

donde

$$S_\tau \left(y, y'_i\right) = \text{ la cantidad de píxeles } p \text{ tal que } \left|y\left(p\right) - y'_i\left(p\right)\right| \leq \tau$$

y τ es un umbral dado.

Algoritmo 3.2 Algoritmo para detectar la oclusión de la imagen y

Require: y la imagen de test con oclusión, A el diccionario de caras, $F_1, F_2, \ldots, F_n$ diferentes conjuntos de píxeles y τ un umbral

1. **for** $i = 1 \ldots n$ **do**
2. Obtenemos y'_i mediante el algoritmo 3.1 y F_i
3. Sea $d_i = |y - y'_i|$
4. Sea $u_i = umbral\left(d_i, \tau\right)$
5. **end for**
6. Sea $max = \arg \max_i S_\tau \left(y, y'_i\right)$
7. Sea F_{n+1} el conjunto de píxeles nulos de u_{max}.
8. Obtenemos y'_{n+1} mediante el algoritmo 3.1 y F_{n+1}
9. Sea $d_{n+1} = \left|y - y'_{n+1}\right|$
10. **return** $u_{n+1} = umbral\left(d_{n+1}, \tau\right)$

El algortitmo 3.2 es una simulación de Monte Carlo. El método de Monte Carlo es un método no determinístico usado para aproximar soluciones matemáticas complejas o costosas de evaluar con exactitud. Esta clase de algoritmos define un dominio $\mathbb{D}$ de las posibles entradas (en nuestro caso, los conjuntos de píxeles) y toma n entradas al azar utilizando una función de distribución. Luego, para cada una de estas entradas se efectua un cálculo determinístico. Por último, se utiliza el resultado de cada cálculo individual (para cada entrada) para formar un resultado final.

Concretamente, el algoritmo 3.2 requiere de un diccionario de caras A, múltiples conjuntos de píxeles $F_1, F_2, \ldots, F_n$ generados tomando bloques de píxeles en forma aleatoria y/o utilizando algunos predefinidos que resulten convenientes para los tipos de oclusión más comunes. Utilizando esto y un umbral

τ (cuyo valor puede ser obtenido en forma experimental), calcula la zona ocluida de una imagen y.

La idea del algoritmo 3.2 es generar múltiples imágenes y_i' (paso 2) y luego se queda con aquella que tenga la mayor similaridad S_τ (paso 6). Una vez hecho esto ya tenemos una primer detección de la oclusión u_{max}. Si bien uno puede quedarse con ese resultado, hemos observado que si utilizamos dichos píxeles no ocluidos como el conjunto de píxeles de entrada (pasos 7 y 8) el resultado final obtenido resulta mucho más preciso y robusto. Esto puede deberse a que el resultado intermedio u_{max} tiene un bajo porcentaje de píxeles ocluidos y además los píxeles no se encuentran localizados en unas pocas regiones de la imagen, sino que por el contrario están esparcidos por todas las regiones en donde no hay oclusión.

Una vez que se obtienen las secciones ocluidas de la imagen, podemos utilizar el resto para el reconocimiento, utilizando los métodos mencionados anteriormente.

En la siguiente sección exponemos algunas conclusiones acerca de los distintos algoritmos planteados en el presente capítulo.

3.4. Síntesis general

En este capítulo mostramos distintos algoritmos de reconocimiento de caras, todos ellos basados en recientes avances de la teoría de *compressed sensing*. Esta teoría sostiene que una señal dada puede ser representada mediante una combinación lineal esparsa de elementos base (también llamados átomos y/o muestras) almacenados en un diccionario. Llevado esto al reconocimiento de rostros, se construye un diccionario de imágenes con las caras de las personas conocidas y luego dada una imagen de test se intenta obtener una combinación lineal esparsa que la aproxime.

En el comienzo de este capítulo se mostró que dicha descomposición esparsa puede ser obtenida resolviendo un problema de programación lineal. Luego, a lo largo del capítulo, se fueron mejorando los sistemas de ecuaciones para obtener dicha combinación lineal esparsa. La primera modificación, correspondiente a la sección 3.1.1, tuvo por objetivo hacer el algoritmo robusto a pequeñas cantidades de ruido gaussiano. Además de esto, se comentaron algunas de las múltiples maneras de extraer *features* de los rostros para reducir la dimensión de las imágenes y poder de esa manera satisfacer las condiciones del sistema de ecuaciones utilizado para conseguir la combinación lineal esparsa.

Posteriormente, se extendió el problema al reconocimiento de rostros aún cuando estos se encontraban parcialmente ocluidos. Esto requirió reformular nuevamente la forma de obtener la combinación lineal esparsa, para modelar errores de una magnitud considerable. Sin embargo, esta formulación sólo funciona correctamente cuando sólo una pequeña fracción de la imagen se encuentra ocluida.

A fin de reconocer imágenes con una mayor proporción de oclusión, se propusieron dos variantes. La primera fue dividir las imágenes en secciones más pequeñas, reconocer cada una de ellas y luego consolidar estos resultados mediante una votación. Si bien esta variante mejora la tasa de reconocimiento cuando las imágenes a reconocer tienen un porcentaje considerable ocluido, tiene el inconveniente que requiere una forma de subdividir las imágenes que logre separar las partes ocluidas de las que no lo están.

Por último, se presentó una segunda variante, la cual fue propuesta en este trabajo, que consiste en detectar las zonas ocluidas de la imagen para luego excluirlas en la etapa de la detección. Este método ha mostrado lograr excelentes resultados.

En el siguiente capítulo, incluimos todos los resultados obtenidos utilizando los diferentes algoritmos.

Capítulo 4

Resultados

En el presente capítulo presentamos los resultados obtenidos al utilizar los algoritmos de detección y reconocimiento con las bases de datos de caras *UHC Face*, *Extended YALE B*[23] y *AR Database*[35].

En primer lugar nos enfocamos en la detección y entrenamos distintos detectores utilizando diferentes parámetros, como ser, si se utiliza una cascada de clasificadores o un árbol de los mismos, o si se utilizan stumps o CART como clasificadores débiles, entre otros. Una vez hecho esto, evaluamos la tasa de detección, la tasa de falsos positivos y generamos curvas ROC para cada uno de ellos a fin de comparar el efecto de los diferentes parámetros.

En segundo lugar probamos los algoritmos de reconocimiento en diversas condiciones: sin oclusión, con anteojos oscuros, con bufanda y cambios de iluminación entre otros.

Por último se presentan las pruebas de integración de los algoritmos de detección y reconocimiento, en donde combinamos ambos métodos a fin de obtener un sistema que no requiera de ningún tipo de intervención manual.

4.1. Bases de datos de caras

Existen diversas bases de datos de caras que nos permiten medir la efectividad de los distintos algoritmos tanto de detección como de reconocimiento. Las bases de datos tienen diferentes características, algunas fueron capturadas en ambientes controlados y otras en ambientes reales. Lo mismo puede ocurrir con la iluminación, escala, tamaño, expresiones y/o oclusión.

Las bases de datos que utilizamos para nuestras mediciones son las siguientes:

1. ***UHC Face***: consta de 142 imágenes en escala de grises, cada una

de ellas posee una o más caras y todas fueron tomadas en ambientes reales. Las imágenes de esta base de datos son de diferentes tamaños y las caras se encuentran en distintas posiciones y escalas. Por todo esto, esta base de datos resulta útil para medir la efectividad de algoritmos de detección de caras.

2. ***Extended YALE B***: consiste en más de 2.414 imágenes en escala de grises de caras frontales correspondientes a 38 individuos. Las imágenes de esta base de datos fueron obtenidas en un ambiente controlado en diversas condiciones de iluminación y luego fueron normalizadas y su tamaño ajustado a 192 x 168 píxeles.

3. ***AR Database***: consta de más de 4000 imágenes a color correspondientes a 126 individuos. Cada individuo posee dos conjuntos de 13 imágenes cada uno, las cuales fueron obtenidas en un ambiente controlado. Estos conjuntos fueron capturados en momentos diferentes con al menos un año de diferencia entre ellos. Cada conjunto de imágenes contiene a su vez distintas expresiones, iluminaciones y algunas imágenes se encuentran parcialmente ocluidas por anteojos negros y/o bufandas.

4. ***Custom Database***: es una base de datos compuesta por alrededor de 10000 imágenes de caras de 24x24 píxeles y cerca de 72000 imágenes de nocaras (imágenes que no poseen caras) de 640x480 píxeles que se usan para generar imágenes de nocaras de 24x24 píxeles. Está formada por imágenes de distintas bases de datos de caras e imágenes de distintas fuentes de internet.

4.2. Detección

En la presente sección se presentan las pruebas realizadas con el algoritmo de detección de rostros visto en el capítulo 2. Para llevar a cabo estas pruebas se utiliza una librería de código abierto de visión por computadora en tiempo real denominada OpenCV (*Open Source Computer Vision*). OpenCV incluye un conjunto de funciones desarrolladas específicamente para el aprendizaje y clasificación de objetos. Implementa una versión del algoritmo de detección de rostros propuesto por Viola y Jones, y las mejoras que fuimos mencionando en el capítulo 2. Además incluye herramientas para crear muestras de entrenamiento y evaluar la performance del detector entrenado, que hemos utilizado en nuestras mediciones en forma complementaria a nuestras propias herramientas.

Se han realizado distintos entrenamientos, en los cuales se fueron variando parámetros que permiten ajustar el detector para obtener mejores resultados.

En todos los entrenamientos, se empleó la base de datos de caras *Custom Database*. Para evaluar la performance de los detectores obtenidos, se han utilizado otras bases, distintas a la de entrenamiento.

La implementación de OpenCV para el entrenamiento del detector de caras, se denomina HaarTrainning. Uno de los primeros parámetros que se debe especificar es la cantidad de muestras positivas y negativas que se usan para entrenar cada etapa de la cascada. Antes de comenzar a entrenar cada clasificador fuerte, el algoritmo se asegura de contar con la cantidad de imágenes especificada. Estas imágenes deben haber "pasado" las etapas anteriores de la cascada, por lo que a medida que se avanza en la cascada, es cada vez más difícil encontrar imágenes negativas. La cantidad de imágenes que usamos en todos nuestros entrenamientos es de 4500 muestras positivas y 3000 muestras negativas por etapa, salvo excepciones que mencionamos explícitamente.

Otro de los parámetros que se debe especificar, es el número de etapas a entrenar. En nuestras pruebas utilizamos una cantidad de 30 etapas. Esta cantidad debe decidirse en conjunto con la tasa de detección (DR_i) y la tasa de falsos positivos (FPR_i) de cada etapa i, ya que el DR y FPR general de la cascada dependen de estos tres parámetros. En nuestros entrenamientos, fijamos DR_i en $99,9\,\%$ y FPR_i en $60\,\%$. Teniendo en cuenta que la cascada constará de 30 etapas, la tasa de detección general será aproximadamente $0,999^{30} \approx 0,9704 = 97,04\,\%$ y la tasa de falsos positivos será aproximadamente $0,60^{30} \approx 0,000000221 = 0,0000221\,\%$.

Otro de los parámetros es el número de *features* a usar en cada clasificador débil. Esto determina el uso de *Stump* vs *CART* como clasificador débil (ver sección 2.2.4). Utilizando una base de datos compuesta por 5000 muestras positivas y 5000 negativas, hemos realizado mediciones de performance variando la cantidad de *features* por clasificador débil. La figura 4.1 muestra la curva ROC (*Receiver Operating Curve*) obtenida con las configuraciones de uno, dos, tres y cuatro *features*. Cada curva se ha generado en base al primer clasificador de la cascada entrenada y variando su respectivo umbral. Como puede verse en la figura, no se obtuvo demasiada diferencia entre los clasificadores que utilizan *CART*, pero si hubo una diferencia más notoria entre éstos y los clasificadores basados en *Stump*. Entre los que usan *CART*, los de dos y cuatro *features* resultaron levemente superiores.

En la figura 4.2, se muestra la cantidad de *features* que el algoritmo de entrenamiento utilizó en cada etapa para los entrenamientos con *Stump* y *CART* de dos, tres y cuatro *features*. Se puede observar que la cantidad aumenta de etapa a etapa pero que se mantiene muy similar entre los diferentes entrenamientos. Esto es porque, si bien los *CART* requieren más *features* por cada clasificador débil, también requieren menor cantidad de clasificadores

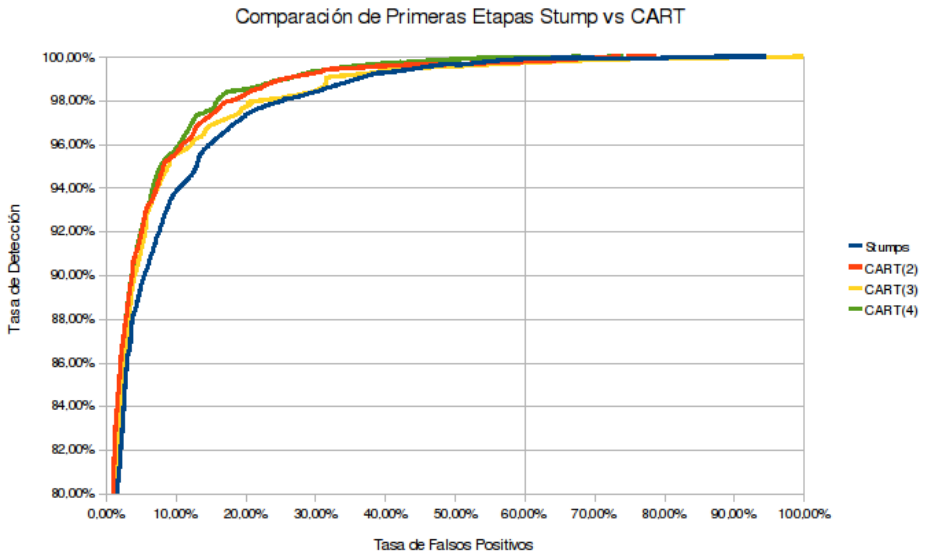


Figura 4.1: Tasa de detección obtenida utilizando *Stump* (un *feature*) y *CART* de dos, tres y cuatro *features* como clasificadores débiles, medida variando el umbral del primer clasificador obtenido en cada entrenamiento.

débiles para realizar la misma tarea.

Como mencionamos antes, OpenCV incluye, junto con el algoritmo de Haar-Trainning, una herramienta para medir la performance de los detectores entrenados. Esta herramienta recibe como entrada un clasificador entrenado y un conjunto de imágenes de prueba con la información de las posiciones y tamaños de las caras presentes en ellas, que toma como "verdad". Luego compara las caras detectadas por el clasificador con la información provista, obteniendo la cantidad de aciertos (caras detectadas), fallos (caras no detectadas) y falsas detecciones (detecciones que no son caras). Con estos datos se puede calcular la tasa de detección y la tasa de falsos positivos.

Usualmente se obtienen múltiples detecciones en la zona de una cara. En estos casos, algunas detecciones poseen una ubicación y escala diferentes a la cara verdadera. Para obtener la detección definitiva se "mezclan" las detecciones de una zona en una sola detección promediándolas. Como se ha visto en [30], se pueden generar curvas ROC variando la cantidad requerida de detecciones para obtener una detección definitiva. La herramienta de medición de performance de OpenCV genera este tipo de curvas. En la figura 4.3 podemos ver el ROC obtenido para los mismos clasificadores de la figura 4.1. Si bien aquí la diferencia de performance ya no es tan clara, el clasificador

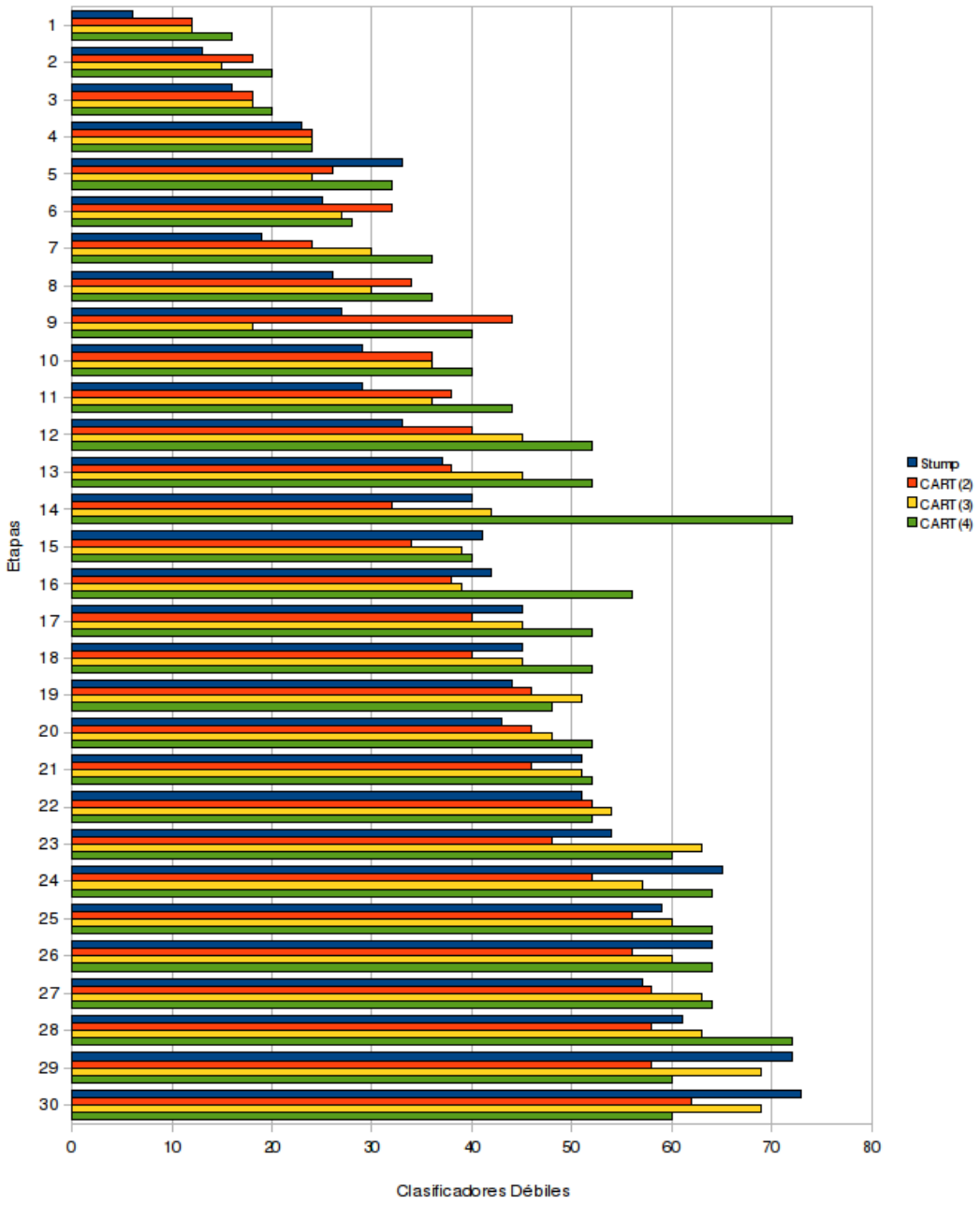


Figura 4.2: Cantidad de *features* utilizados en cada etapa de la cascada en los entrenamientos con *Stump* y *CART* de dos, tres y cuatro *features*.

que utiliza *CART* de dos *features* parece ser levemente mejor.

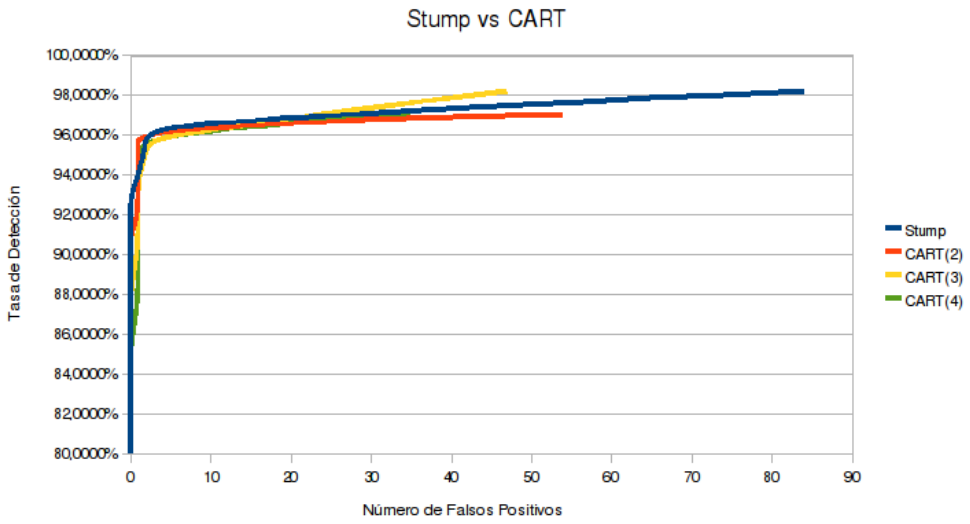


Figura 4.3: Tasa de detección obtenida utilizando *Stump* (un *feature*) y *CART* de dos, tres y cuatro *features* como clasificadores débiles, medida variando la cantidad de detecciones requeridas por cada detección definitiva para ser considerada una detección.

Una de las mejoras vistas sobre el algoritmo de Viola y Jones es el uso de una estructura de árbol en lugar de la cascada de clasificadores fuertes (ver sección 2.2.5). La implementación de OpenCV permite realizar el entrenamiento utilizando esta nueva estructura. Para ello se debe especificar la cantidad máxima de ramificaciones del árbol. Si este valor es cero, el árbol entrenado resulta ser una cascada. Además, se debe especificar la cantidad mínima de muestras positivas, por cada cluster del árbol, necesarias para generar una ramificación. Los entrenamientos realizados, incluyen árboles de cero, una, dos y tres ramificaciones. En la figura 4.4 se presentan los resultados de aplicar los clasificadores entrenados a la base de datos *UHC Face*. Allí se puede observar que el árbol que mayor tasa de detección tiene es el de una ramificación. Las mediciones fueron generadas variando la cantidad de detecciones requeridas por cada detección definitiva.

El siguiente experimento que hicimos fue variar el conjunto de *features* utilizado en el entrenamiento. Como vimos en sección 2.2.1, además de los *features* propuestos originalmente por Viola y Jones, se han propuesto un conjunto de *features* nuevos que incluyen *features* rotados. La implementación de OpenCV permite seleccionar entre tres conjuntos de *features* a utilizar en el entrenamiento. Estos son: *BASIC*, que son los *features* propuestos por Viola y Jones, *CORE*, que son todos los *features* propuestos sin inclinación

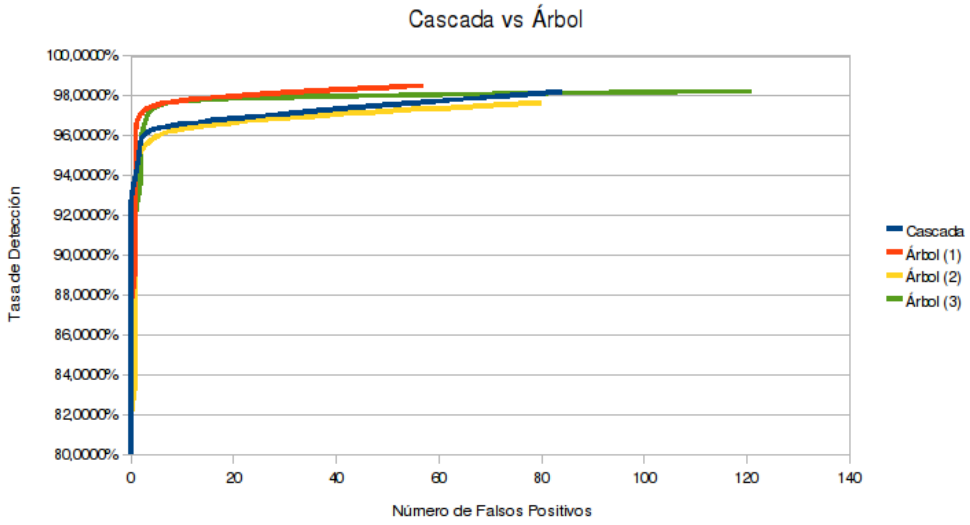


Figura 4.4: Tasa de detección obtenida con la base de datos *UHC Face* para árboles de clasificadores de cero, una, dos y tres ramificaciones, medida variando la cantidad de detecciones requeridas por cada detección definitiva para ser considerada una detección.

y *ALL*, que incluye todos los *features* incluso los rotados. En la figura 4.5 se observan las curvas ROC obtenidas para los tres conjuntos. Los mejores resultados se obtuvieron con el entrenamiento que utilizó el conjunto completo.

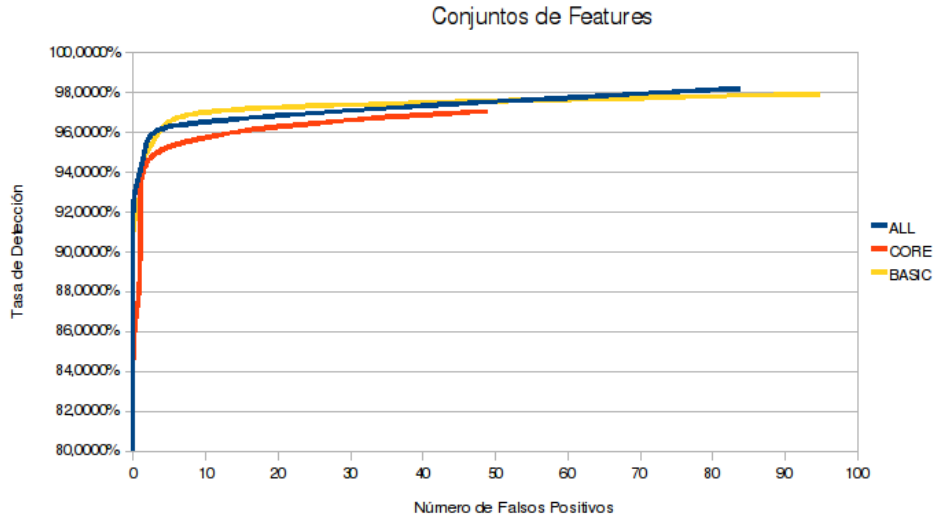


Figura 4.5: Tasa de detección obtenida con la base de datos *UHC Face* para distintos conjuntos de *features*. BASIC → los *features* propuestos por Viola y Jones, CORE → los *features* básicos y extendidos sin inclinación, ALL → los *features* básicos, extendidos y rotados.

Finalmente, realizamos una comparación entre uno de los detectores entrenados en estas pruebas y uno de los detectores distribuidos con OpenCV para caras frontales (haarcascade_frontalface_alt2). Los resultados obtenidos fueron muy similares. Con la base de datos *UHC Face*, nuestro detector mostró un desempeño levemente mejor, con una tasa de detección similar pero con menos falsos positivos. Esta diferencia se debe principalmente a que el detector de OpenCV posee 20 etapas contra las 30 etapas de nuestro detector. En la figura 4.6 se observan los ROC de ambos entrenamientos.

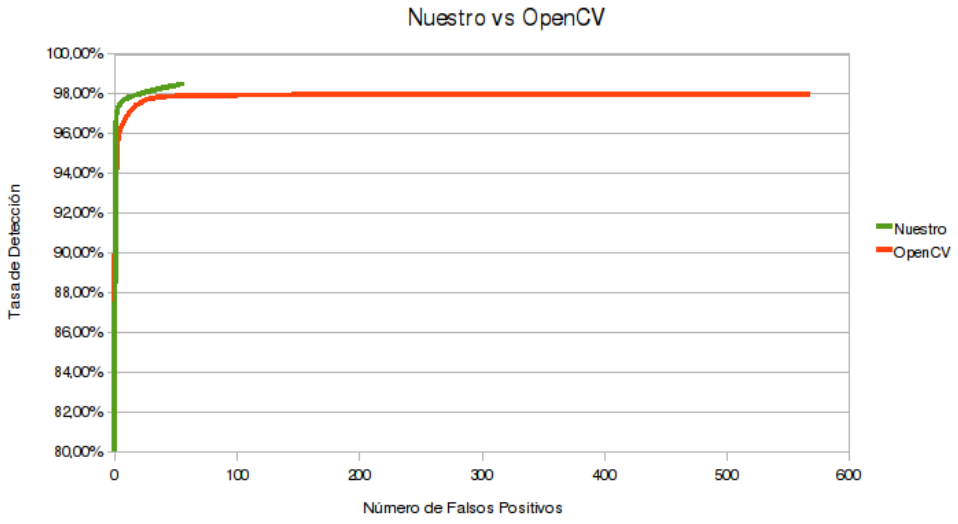


Figura 4.6: Comparación del detector entrenado en las pruebas y uno de los detectores incluidos en OpenCV.

En la tabla 4.1, se muestran los resultados obtenidos con las distintas pruebas de entrenamiento realizadas. Allí se puede comparar la cantidad de caras detectadas, la cantidad de caras no detectadas y los falsos positivos que produjo cada detector al emplear la base de datos *UHC Face*. Los porcentajes que se muestran en la tabla, incluyendo el porcentaje de falsos positivos, se calculan respecto al total de caras presentes en las imágenes de prueba. Como puede verse, todos los detectores son muy buenos y no presentan diferencias importantes en los resultados.

A modo de conclusión, podemos decir que durante las pruebas realizadas obtuvimos detectores muy eficientes. A medida que fuimos incorporando al entrenamiento las diferentes mejoras, pudimos notar leves cambios en el desempeño del detector, muchas veces mejorando la detección. Los resultados de todos los entrenamientos fueron muy buenos, obteniendo tasas de detec-

Entrenamiento	Det.	%	No Det.	%	FP	%
STUMP	337	98,25 %	6	1,75 %	84	24,49 %
CART(2)	333	97,08 %	10	2,92 %	54	15,74 %
CART(3)	337	98,25 %	6	1,75 %	47	13,70 %
CART(4)	333	97,08 %	10	2,92 %	35	10,20 %
CASCADA	337	98,25 %	6	1,75 %	84	24,49 %
ARBOL(1)	338	98,54 %	5	1,46 %	57	16,62 %
ARBOL(2)	335	97,67 %	8	2,33 %	80	23,32 %
ARBOL(3)	337	98,25 %	6	1,75 %	121	35,28 %
BASIC	336	97,96 %	7	2,04 %	95	27,70 %
CORE	333	97,08 %	10	2,92 %	49	14,29 %
ALL	337	98,25 %	6	1,75 %	84	24,49 %
OpenCV	336	97,96 %	7	2,04 %	569	165,89 %

Tabla 4.1: Tabla comparativa de los diferentes detectores entrenados

ción muy elevadas. Dado que en algunos casos la diferencia entre dos entrenamientos es de tan solo una cara detectada, es difícil concluir que ciertos cambios son buenos o contraproducentes con tan buenos resultados. Además, es posible que no se esté empleando el método más adecuado para medir la performance, o incluso, el uso de una base de datos poco representativa de la "realidad".

En primer lugar, utilizamos para medir la performance la primer etapa de cada clasificador entrenado. Si bien es de esperar que el primer clasificador marque una tendencia y que el resto de los clasificadores posean un desempeño similar al primero, no necesariamente esto debe ser así, y puede suceder que una estructura de clasificadores obtenga mejores resultados que otra en las etapas finales. Por otro lado este método no es aplicable para comparar distintos entrenamientos de árboles ya que éstos no varían en la primera etapa. Otro de los métodos usados es generar el ROC variando la cantidad de detecciones requeridas para considerar una detección. Este último método no cuenta con un buen sustento teórico ni tampoco en nuestras pruebas fue capaz de demostrar con certeza las mejoras propuestas.

En la sección de pruebas integradoras (4.4), se presentarán más resultados de detección utilizando otras bases de datos e incluyendo imágenes con oclusión.

4.3. Reconocimiento

En esta sección detallamos los resultados obtenidos con los algoritmos de reconocimiento de caras del capítulo 3. Podemos dividir las pruebas realizadas en aquellas en las que las imágenes a reconocer no están ocluidas y aquellas en las que hay una porción ocluida en las imágenes (algoritmos de las secciones 3.1.1 y 3.3 respectivamente).

4.3.1. Sin oclusión

En las pruebas de reconocimiento sin oclusión se utilizaron dos bases de datos de caras, la *Extended YALE B* y la *AR Database*. Para cada una de las bases de datos de caras se realizaron diversos experimentos para observar el efecto de los parámetros ϵ (ver sección 3.1.1), la distribución aleatoria utilizada para generar la matriz R (ver sección 3.1.2) y por último, el tamaño d del espacio de características (que resulta de la cantidad de filas de la matriz R).

4.3.1.1. Resultados usando *Extended Yale B*

Como ya mencionamos antes, la base de datos de caras *Extended Yale B* consta de 38 sujetos. De cada uno de estos sujetos se eligieron 22 imágenes para el diccionario de caras y 21 como imágenes de test. Esta selección fue realizada en forma aleatoria para no presuponer ningún tipo de iluminación favorable.

Esta base de datos tiene la característica de que, debido al método que se utilizó para adquirir las imágenes, todas las imágenes se encuentran perfectamente alineadas. Esta característica de esta base de datos sin duda influirá positivamente en la tasa de reconocimiento, algo a tener en cuenta si se quiere comparar los resultados con otras bases de datos.

La primer medición que realizamos fue variar el valor de ϵ y observar la tasa de reconocimiento del algoritmo. Como puede observarse en la figura 4.7, la tasa de reconocimiento fue del 100 % cuando el valor de ϵ toma los valores $0,01$, $0,05$ y $0,1$. También puede verse que el algoritmo no es muy estable con valores más pequeños, en este caso con $0,001$.

La siguiente prueba que realizamos fue comparar la tasa de reconocimiento obtenida utilizando $\epsilon = 0{,}05$ y generando la matriz R con diferentes distribuciones aleatorias. En nuestra prueba utilizamos tres distribuciones aleatorias: Uniforme [-1, 1], Uniforme [0, 1], Gaussiana ($\mu = 0, \sigma = 1$). Con esta base de datos de caras el reconocimiento obtenido no presentó ninguna variación y en todos los casos fue del 100 %.

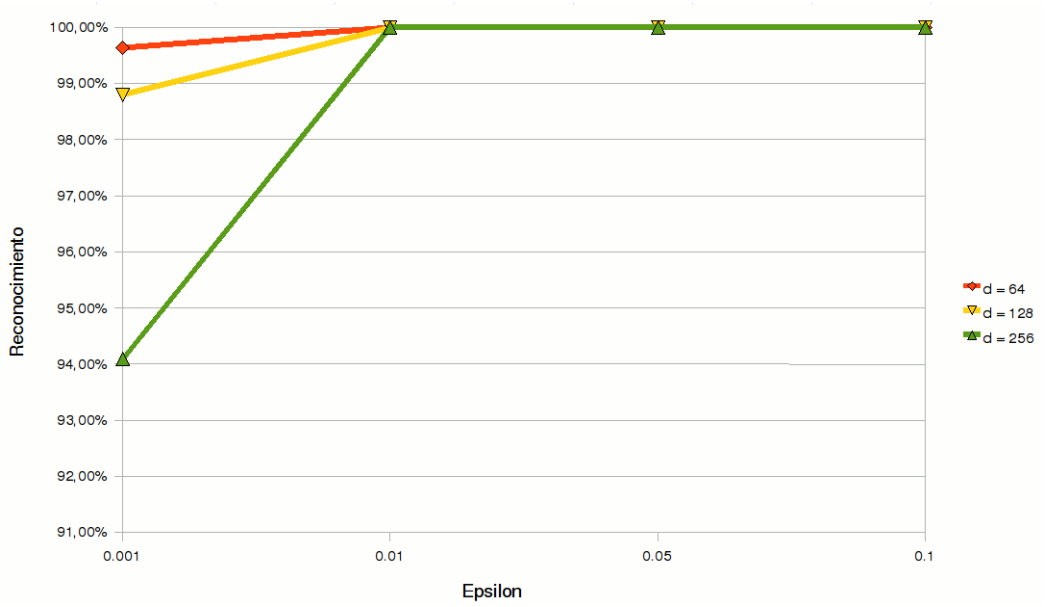


Figura 4.7: Tasa de reconocimiento obtenida utilizando la base de datos de caras *Extended Yale B* y variando el valor de ϵ. Las diferentes curvas se corresponden con distintos tamaños de d

La tercer prueba consiste en medir la tasa de reconocimiento variando el tamaño d del espacio de características (recordemos que $R \in \mathbb{R}^{d \times m}$ donde m es la cantidad de píxeles que tiene cada imagen del diccionario). La figura 4.8 muestra los resultados obtenidos para diferentes valores de d utilizando $\epsilon = 0{,}05$.

En la figura 4.8 se puede observar que se producen algunos fallos cuando el espacio de características es pequeño ($d = 32$) y que al utilizar más características, el reconocimiento mejora. Sin embargo, también puede observarse que cuando $d = 512$ comienzan a aparecer algunos fallos. Esto se debe a que a medida que d aumenta, la matriz $\tilde{A} = RA \in \mathbb{R}^{d \times n}$ se acerca cada vez más a una matriz cuadrada. Cuando esto sucede, cada vez hay menos soluciones posibles para el sistema de ecuaciones 3.13 ($\tilde{y} = \tilde{A}x \in \mathbb{R}^d$), las soluciones obtenidas dejarán de ser esparsas y por lo tanto, el reconocimiento no será satisfactorio. Teniendo en cuenta que el diccionario tiene $n = 871$ imágenes, era de esperar que con $d = 512$ comience a fallar.

Por último, en la figura 4.9 se muestran los fallos obtenidos en función del ϵ y de d. Como puede apreciarse en la misma, cuando d es grande en comparación con n, el valor de ϵ cobra mucha relevancia.

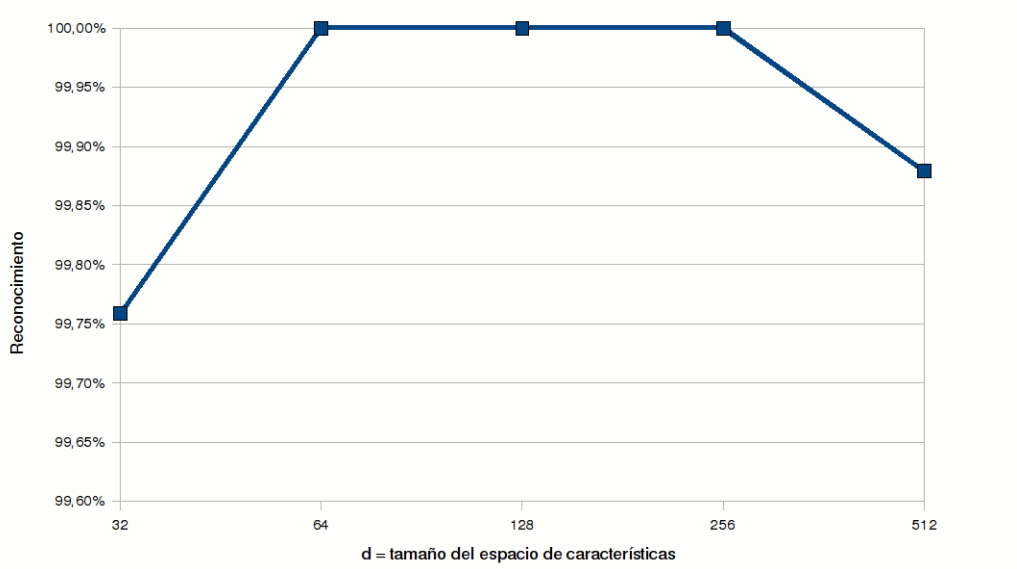


Figura 4.8: Tasa de reconocimiento obtenida utilizando la base de datos de caras *Extended Yale B* y variando el valor de d. En esta ocasión, $n = 871$ imágenes fueron utilizadas en el diccionario y 829 como imágenes de test.

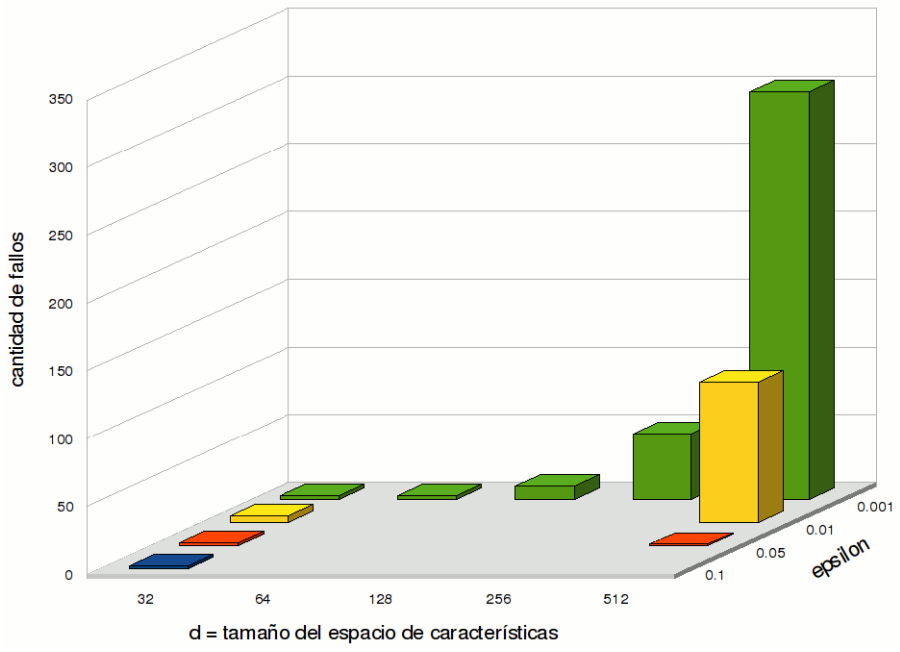


Figura 4.9: Cantidad de fallos obtenidos utilizando la base de datos de caras *Extended Yale B* para distintos ϵ y d. La cantidad de fallos está expresada sobre un total de 829 imágenes de test y utilizando $n = 871$ imágenes para el diccionario de caras.

4.3.1.2. Resultados usando *AR Database*

Para las pruebas sin oclusión, con imágenes de *AR Database*, se seleccionaron 60 individuos. De cada individuo se utilizaron 7 imágenes no ocluidas del primer conjunto para el diccionario de caras y otras 7 del segundo conjunto para test.

Es importante destacar que las pruebas realizadas con esta base de datos de caras son mucho más exigentes que las de *Extended Yale B*. Esto se debe a varios factores, el primero de ellos radica en que, a diferencia de la *Extended Yale B*, en la base *AR Database* las imágenes de los individuos no sólo tienen diferentes iluminaciones sino también diferentes expresiones. Además, las imágenes utilizadas para test fueron tomadas al menos un año después que las que se utilizan en el diccionario de caras del algoritmo. Otro aspecto que hace que esta prueba sea más exigente que la anterior es que, si bien las imágenes fueron recortadas y alineadas manualmente, la alineación entre las distintas imágenes no es tan precisa como en la *Extended Yale B*, ya que durante la captura de las imágenes los sujetos fotografiados debieron moverse para colocarse anteojos, bufandas y hacer las diferentes expresiones. Por último, hay que tener en cuenta que en esta prueba sólo disponemos de 7 imágenes por individuo en el diccionario de caras, respecto de las 22 de la prueba anterior y por lo tanto el espacio de caras resultante es mucho más acotado.

La primer medición realizada es medir la influencia de ϵ en el reconocimiento. La figura 4.10 muestra la tasa de reconocimiento obtenida variando ϵ y fijando $d = 256$, y generando la matriz aleatoria $R \in \mathbb{R}^{d \times m}$ con una distribución gaussiana ($\mu = 0, \sigma = 1$). Como puede observarse en la figura, la tasa de reconocimiento alcanza valor más alto con $\epsilon = 0,05$. Esto se debe a que a medida que ϵ es más grande, el algoritmo es más robusto al ruido; pero si el valor es muy grande, el método deja de ser sensible a algunas características de las personas a reconocer. Algo interesante a destacar es que en [49] también se utilizó $\epsilon = 0,05$ para sus pruebas.

La siguiente prueba que hicimos fue generar la matriz R utilizando diferentes distribuciones aleatorias. Para ello, se fijaron los parámetros $d = 256$ y $\epsilon = 0,05$ y se generaron matrices aleatorias utilizando 3 distribuciones: Uniforme [0, 1], Uniforme [-1, 1], Gaussiana ($\mu = 0, \sigma = 1$).

La figura 4.11 muestra la tasa de reconocimiento obtenida (a) y el tiempo promedio requerido para reconocer una imagen (b). Como puede observarse en las figuras, tanto la distribución Gaussiana ($\mu = 0, \sigma = 1$), como la Uniforme [-1, 1] tuvieron un $94,90\,\%$ de reconocimiento, mientras que la Uniforme [0, 1] tuvo un reconocimiento significativamente menor, $84,22\,\%$.

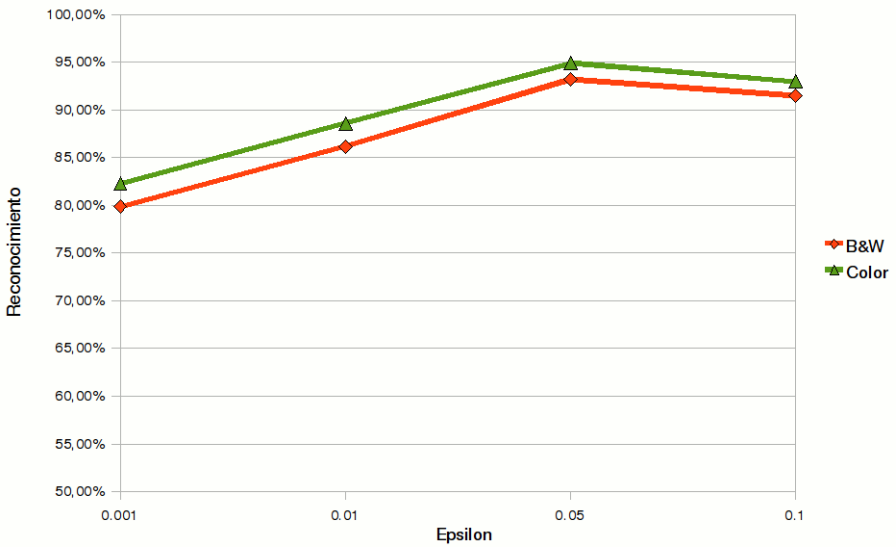


Figura 4.10: Tasa de reconocimiento obtenida utilizando la base de datos de caras *AR Database* y variando el valor de ϵ. Se fijó $d = 256$ y se generó la matriz aleatoria R con una distribución gaussiana ($\mu = 0, \sigma = 1$)

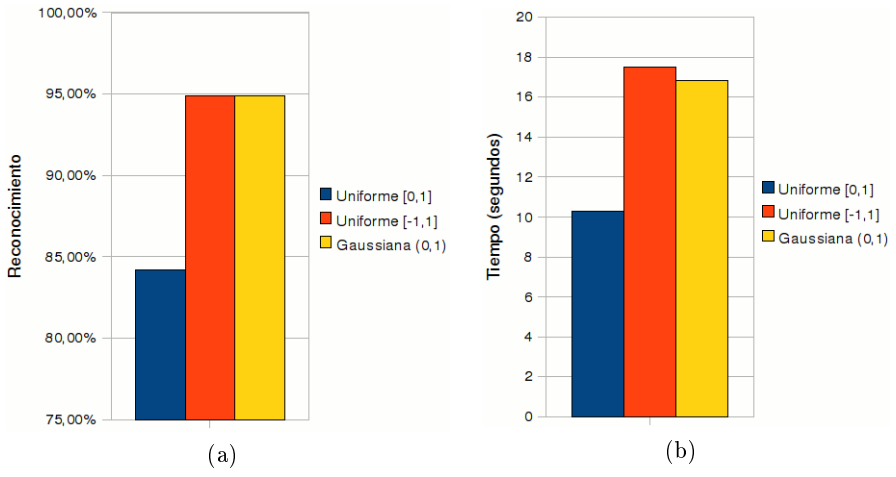


Figura 4.11: **(a)** Tasa de reconocimiento generando la matriz R con diferentes distribuciones aleatorias y usando $d = 256$ y $\epsilon = 0,05$. **(b)** Tiempo promedio requerido para reconocer un rostro.

Si comparamos los tiempos de ejecución entre las dos distribuciones que tuvieron mayor éxito, observamos que la distribución Gaussiana ($\mu = 0, \sigma = 1$) es levemente más veloz que la Uniforme [-1, 1]. Por este motivo, decidimos utilizar en nuestro algoritmo la distribución Gaussiana.

El tercer experimento consiste en variar el tamaño d del espacio de características. A diferencia de las pruebas sin oclusión realizadas con la base de imágenes *Extended Yale B*, con la base *AR Database* no es posible utilizar $d = 512$ ya que solo disponemos de $n = 413$ imágenes en el diccionario de caras (Recordemos que conforme a lo explicado en la sección 3.1.2, debe satisfacerse la condición $d < n$), por este motivo sólo utilizaremos en las mediciones $d = 32$, 64, 128 y 256.

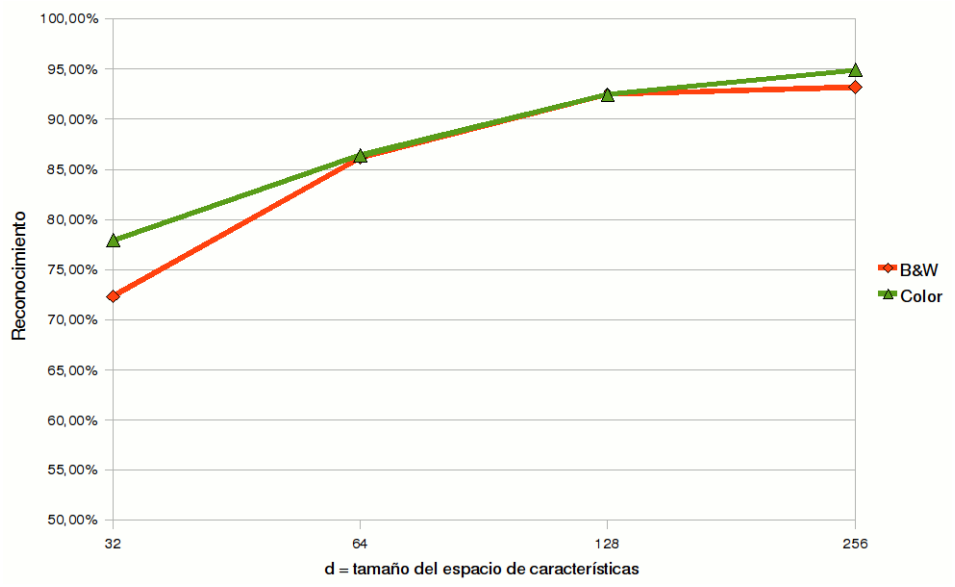
En la figura 4.12(a) podemos ver como aumenta el reconocimiento cuando aumentamos el tamaño del espacio de características. También podemos observar que el reconocimiento mejora levemente cuando se utilizan imágenes a color respecto de las imágenes en escala de grises. Utilizando $d = 256$, obtuvimos $94,90\,\%$ de reconocimientos exitosos.

Por otro lado, la figura 4.12(b) muestra el tiempo promedio necesario para reconocer un rostro en función de d. Como era de esperarse, a medida que aumenta el tamaño del espacio de características el tiempo es mayor. Un dato curioso que resulta de las pruebas realizadas es que las imágenes a color requirieron en promedio menos tiempo que las imágenes en escala de grises. Esto se debe a que cuando la librería que resuelve el sistema de ecuaciones 3.11 no encuentra una solución esparsa (y por ende, el reconocimiento falla), ésta requiere más tiempo que cuando la solución es hallada satisfactoriamente. Por esto, dado que las imágenes color tuvieron una mayor tasa de reconocimiento, el tiempo promedio de reconocimiento fue menor en las imágenes color.

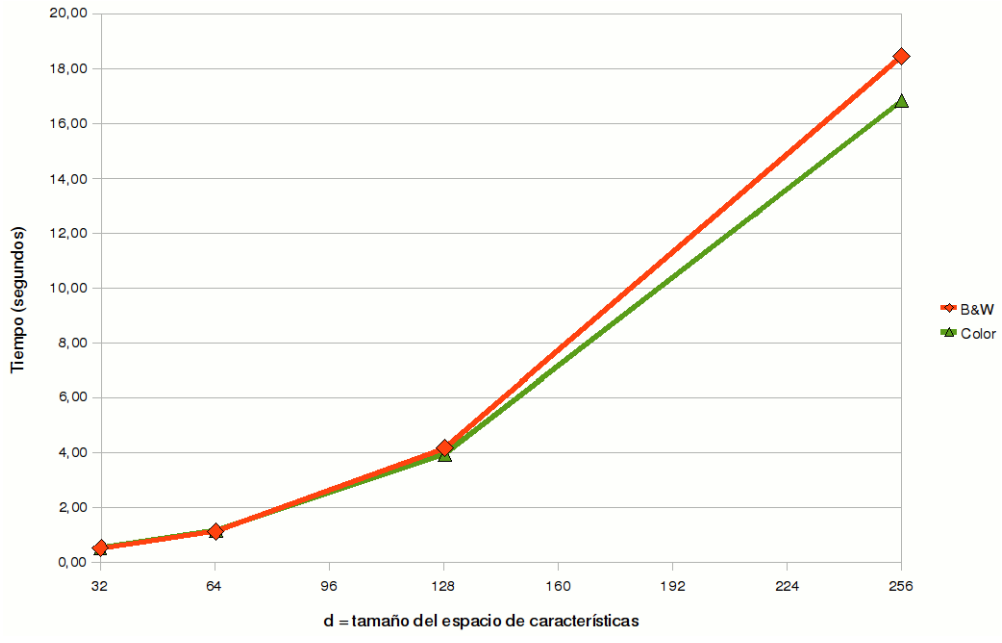
4.3.2. Con oclusión

Para las pruebas con oclusión se utilizó la base de datos de caras *AR Database*. De esta base se seleccionaron 60 individuos y de cada uno de ellos se utilizaron 8 imágenes no ocluidas para el diccionario de caras y 4 imágenes ocluidas para test. De las imágenes de test, 2 se encuentran ocluidas por tener anteojos negros y las otras dos tienen una bufanda. En la figura 4.13 se pueden ver algunas de las imágenes utilizadas.

Podemos dividir las pruebas con oclusión realizadas en función del algoritmo utilizado. En primer lugar, hicimos pruebas utilizando la formulación descripta en la sección 3.3, a la que denominamos "Modelo Simple", que incluye un modelo de error que contempla una porción ocluida en la imagen



(a)



(b)

Figura 4.12: Tasa de reconocimiento **(a)** y tiempo promedio requerido para reconocer un rostro **(b)** obtenido utilizando la base de datos de caras *AR Database* y variando el valor de d. Se fijó $\epsilon = 0,05$ y se generó la matriz aleatoria R con una distribución gaussiana ($\mu = 0, \sigma = 1$)

(a) (b) (c)

Figura 4.13: Algunas de las imágenes de la base de datos de caras *AR Database* que fueron utilizadas para el reconocimiento con oclusión. **(a)** imagen sin ocluir, utilizada en el diccionario de caras. **(b)** imagen con anteojos negros utilizada como imagen de test. **(c)** imagen con bufanda utilizada como imagen de test.

a reconocer. Como veremos más adelante, esta formulación produce buenos resultados siempre y cuando la porción ocluida de la imagen sea pequeña.

En segundo lugar, realizamos mediciones utilizando la formulación descripta en la sección 3.3.1, a la que denominamos "Modelo Particionado" que es un intento por solucionar el problema de grandes porciones ocluidas que presenta la formulación anterior. Como vimos anteriormente, este método consiste en subdividir la imagen a detectar en fragmentos más pequeños, reconocer cada uno de ellos y luego por medio de una votación consolidar los resultados obtenidos.

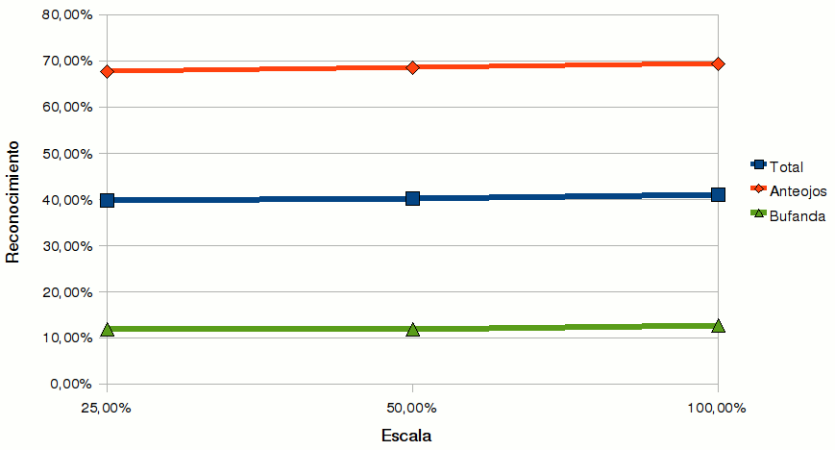
Por último, analizamos los resultados obtenidos utilizando nuestro método descripto en la sección 3.3.2, al que denominamos "Detección de la oclusión", que consiste en detectar qué porción de la imagen se encuentra ocluida y utilizar la información restante para el reconocimiento.

4.3.2.1. Modelo Simple

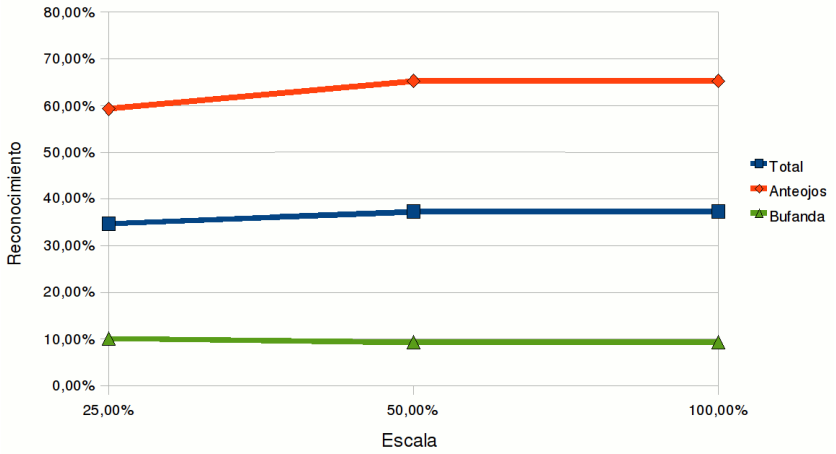
Cuando las imágenes a reconocer tienen oclusión, es necesario utilizar una formulación que contemple la posibilidad de que exista un error de una magnitud arbitraria. La primera, que denominamos "Modelo Simple" se basa en resolver la ecuación $y = [A, I] \begin{bmatrix} x \\ e \end{bmatrix} = Bu$ (para más detalles ver la ecuación 3.19) donde x son coeficientes esparsos y e representa el error cometido.

Dado que utilizando esta formulación no se extraen *features*, sino que por el contrario se aprovecha toda la información de la imagen, decidimos medir cuánto afecta el tamaño de las imágenes para el reconocimiento. Esta prueba la realizamos para imágenes color y en escala de grises.

La figura 4.14 muestra la tasa de reconocimiento obtenida al escalar las imágenes (las del diccionario de caras y las de test) tanto para imágenes a color, como para imágenes a escala de grises (figuras 4.14(a) y 4.14(b) respectivamente). Además, en cada uno de los experimentos, se discriminó la



(a)



(b)

Figura 4.14: Tasa de reconocimiento para imágenes ocluidas utilizando el "Modelo Simple". El experimento se realizó para imágenes color y en escala de grises ((a) y (b) respectivamente) y en cada uno de ellos se midió la performance obtenida variando el tamaño de las imágenes utilizadas. Las diferentes curvas muestran los resultados de imágenes con anteojos negros, bufandas y otra combinando ambas categorías.

tasa de reconocimiento obtenida con imágenes ocluidas por anteojos negros y las ocluidas con bufanda.

Como puede apreciarse en la figura, el reconocimiento de imágenes ocluidas con bufandas fue malo en todos los casos. Esto se debe a que las imágenes ocluidas con bufandas tienen una mayor superficie cubierta que las que tienen anteojos negros y, como explicamos en la sección 3.3, cuando la porción ocluida alcanza el 33 %, esta formulación comienza a fallar [48].

Ahora bien, si analizamos solo las imágenes que tienen anteojos negros, podemos observar que las imágenes a color tuvieron un mayor índice de reconocimientos exitosos (alrededor del 5 %). Además, podemos observar que cuando las imágenes son más chicas, la información del color resulta más importante que con imágenes grandes. Este fenómeno también ocurre en la visión de los seres humanos, donde la información del color resulta de especial importancia cuando las imágenes son pequeñas o de baja calidad [41].

Por último, es interesante graficar el error e que se obtiene en el reconocimiento. La figura 4.15 muestra dos reconocimientos distintos. El primero de ellos busca reconocer una imagen que tiene anteojos negros (a) y el reconocimiento logra reconocer a la persona correcta (b). Luego, se graficó el error e (recordar que $e \in \mathbb{R}^m$, donde $m = ancho \times alto$) como si fuese una imagen. Como puede observarse en la figura 4.15(c), el error se encuentra concentrado en la zona ocluida de la imagen; mas aún, el error toma la forma de los ojos que no están presentes en la imagen a detectar.

El segundo reconocimiento que muestra la figura 4.15, intenta determinar la identidad de una imagen del mismo sujeto, pero esta vez tiene la parte inferior de la imagen ocluida con una bufanda (d). A diferencia del caso anterior, en este caso el reconocimiento falla y devuelve otra persona que no es la esperada (e). Como la persona detectada no es la correcta, el error e obtenido es grande, se encuentra esparcido por toda la imagen y no tiene una forma definida (figura 4.15(f)).

El próximo método que evaluamos busca mejorar el reconocimiento cuando la porción ocluida de la imagen a reconocer es grande (como en el caso de las imágenes ocluidas con bufandas).

4.3.2.2. Modelo Particionado

Este método busca mejorar el caso en el que la imagen a reconocer tiene una porción ocluida considerable y a su vez, la porción ocluida se encuentra concentrada en una zona de la imagen. Tal como explicamos en la sección

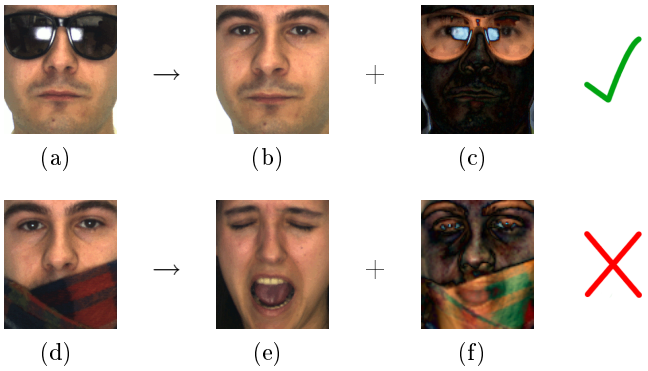


Figura 4.15: Resultados mostrados en forma visual. **(a)** imagen ocluida con anteojos negros utilizada para el reconocimiento. **(b)** el sujeto que fue correctamente detectado. **(c)** el error e mostrado como una imagen. Puede apreciarse cómo el error toma forma de los ojos que fueron ocluidos en la imagen (a). **(d)** imagen de la misma persona que (a), pero ocluida con una bufanda; esta imagen es utilizada para el reconocimiento. **(e)** la persona devuelta por el algoritmo, la cual difiere de la esperada. **(f)** al igual que con (c), se graficó el error e, pero a diferencia de la prueba anterior, el error se encuentra por toda la imagen, sin una forma clara.

3.3.1, este método consiste en dividir la imagen en partes más pequeñas y reconocer cada una de ellas. Luego se hace una votación entre todos los reconocimientos para determinar la persona en cuestión. La figura 4.16 muestra esto en forma gráfica.

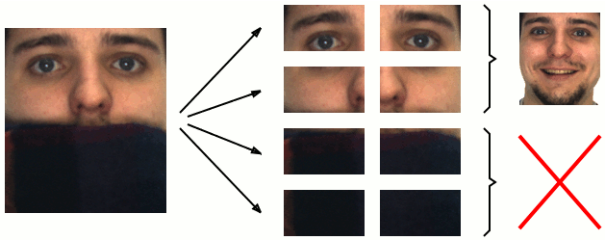


Figura 4.16: El "Modelo Particionado" consiste en dividir la imagen a reconocer en secciones más pequeñas, reconocer cada una de ellas y luego votar entre todos los reconocimientos para determinar la persona en cuestión.

En los experimentos realizados, la imagen se divide en 8 fragmentos de igual tamaño. Se eligió la partición que se muestra en la figura 4.16, es decir que se divide a la mitad verticalmente y luego en 4 en forma horizontal. Esta forma de dividir la imagen fue propuesta en [48] y decidimos respetarla para luego poder contrastar los resultados de este método con el nuestro (cuyos resultados se describen en la siguiente sección).

Las tablas 4.2 y 4.3 muestran la tasa de reconocimiento obtenida utilizando

Algoritmo	**Color**		
	Anteojos	**Bufandas**	**Total**
Modelo Simple	69,49 %	12,71 %	41,10 %
Modelo Particionado	86,44 %	95,76 %	91,10 %

Tabla 4.2: Tabla comparativa con la tasa de reconocimiento obtenida para los distintos algoritmos usando imágenes a color

Algoritmo	**Escala de Grises**		
	Anteojos	**Bufandas**	**Total**
Modelo Simple	65,25 %	9,32 %	37,29 %
Modelo Particionado	78,81 %	90,07 %	86,44 %

Tabla 4.3: Tabla comparativa con la tasa de reconocimiento obtenida para los distintos algoritmos usando imágenes en escala de grises

imágenes a color y en escala de grises respectivamente. Como puede apreciarse en ambas tablas, la tasa de reconocimiento aumenta significativamente con el "Modelo Particionado" respecto del "Modelo Simple". Esto demuestra, que la idea de particionar el problema en imágenes más pequeñas soluciona el problema de tener un gran porcentaje de la imagen ocluida, ya que las zonas que tienen un alto grado de oclusión quedan descartadas por tener un SCI bajo (ver sección 3.2) y las otras sirven para reconocer la identidad del individuo.

Otro aspecto interesante es que con el "Modelo Particionado" la tasa de reconocimiento de imágenes ocluidas con bufandas es superior a la de los anteojos (contrariamente a lo que pasa con el "Modelo Simple"). Esto se debe a que la zona de los ojos contiene más información acerca de la identidad del individuo que la zona de la boca [41][48].

Tal como comentamos en la sección 3.3, el principal problema del "Modelo Particionado" es obtener una partición que permita aislar las zonas ocluidas de aquellas que no lo están. La partición aquí utilizada, fue propuesta en trabajos que también utilizaban la base de datos de caras *AR Database*; es decir, que quienes idearon esta forma de subdividir las imágenes, lo hicieron a sabiendas del tipo de oclusión que esperaban tener. Es por ello que no resulta sorprendente que la partición se adecúe tan bien a los tipos de oclusión presentes en esta base de datos de caras.

En la siguiente sección, mostramos los resultados de nuestro método, que permite detectar las zonas ocluidas para luego descartarlas en el reconoci-

miento, evitando de esta manera la necesidad de tener una partición fija.

4.3.2.3. Detección de la oclusión

Nuestro método consta de dos etapas. La primera de ellas consiste en detectar la oclusión y la segunda en reconocer al individuo en cuestión utilizando únicamente las zonas no ocluidas. Si bien a nosotros nos interesa el resultado de la segunda etapa, el éxito de ésta está sujeto al de la primer etapa.

En esta sección no evaluaremos la performance de cada etapa individualmente, sino la del conjunto de ambas etapas. Dicho en otras palabras, consideraremos un caso exitoso cuando una imagen parcialmente ocluida sea correctamente reconocida por el algoritmo.

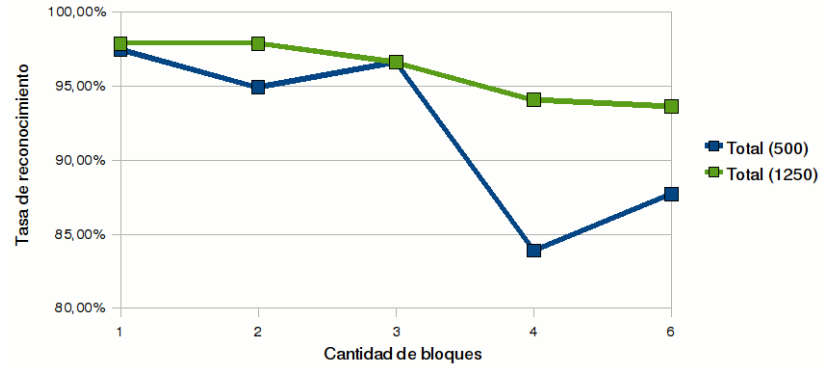
Este método tiene varios parámetros. Entre ellos se encuentra el umbral τ, la cantidad de píxeles de los conjuntos F_i que serán utilizados en el algoritmo 3.2 de la sección 3.3.2 y la forma de obtener dichos conjuntos.

En la sección 3.3.2 presentamos cuatro métodos para obtener dichos conjuntos, de los cuales vimos que sólo dos de ellos se adecúan a nuestro caso. Estos dos métodos son "Utilizar conjuntos de píxeles predefinidos, teniendo en cuenta las oclusiones más comunes que pueden presentarse" y "Tomar bloques de píxeles contiguos en forma aleatoria". El primero de estos no lo utilizamos para nuestras mediciones para no otorgarle al detector ningún conocimiento previo acerca de la oclusión existente en la base de datos de caras. El segundo método requiere de otro parámetro que es la cantidad de bloques de píxeles a elegir.

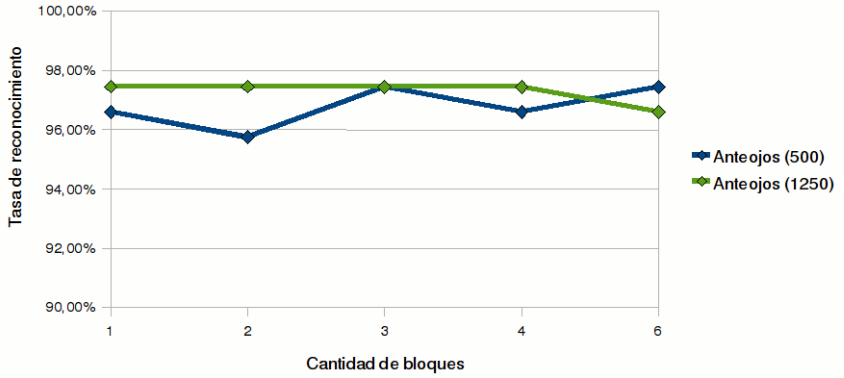
Nuestro primer experimento entonces fue fijar el umbral $\tau = 20$ y variar la cantidad de bloques de píxeles. Este experimento lo hicimos utilizando conjuntos de 500 píxeles ($6,25\,\%$) y otros utilizando 1250 píxeles ($15,5\,\%$); sobre un total de 8064 píxeles.

La figura 4.17 muestra la tasa de reconocimiento obtenida utilizando diferentes cantidades de bloques de píxeles contiguos. Las curvas color azul representan las pruebas que se hicieron tomando un total de 500 píxeles aproximadamente; mientras que las curvas color verde representan pruebas en las que los bloques contienen alrededor de 1250 píxeles.

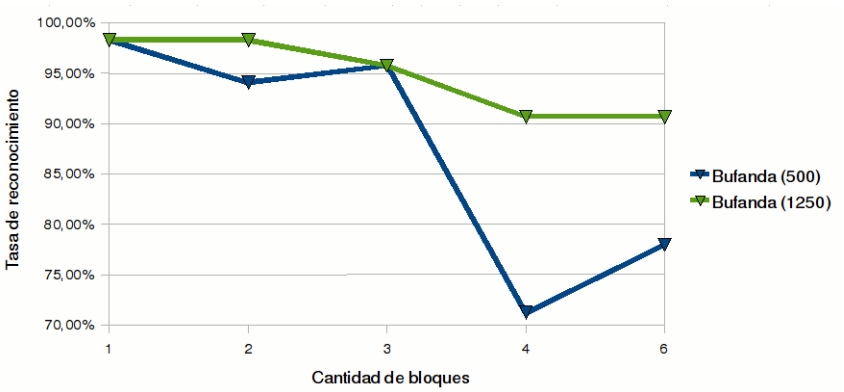
Como puede observarse, a medida que la cantidad de bloques aumenta, la tasa de reconocimiento de las bufandas comienza a decaer (figura 4.17(c)). Esto se debe a que al haber más bloques, la probabilidad de que todos los bloques se encuentren en una zona no ocluida es mucho menor. Además



(a)



(b)



(c)

Figura 4.17: Tasa de reconocimiento obtenida utilizando el método de detectar la oclusión, para luego excluir dicha porción. En estas pruebas se fijó el umbral $\tau = 20$ y se varió la cantidad de bloques de píxeles contiguos elegidos para los conjuntos F_i del algoritmo 3.2. Las curvas color azul son pruebas en donde los conjuntos F_i tienen aproximadamente 500 píxeles, mientras que las curvas color verde tienen 1250 píxeles en dichos conjuntos. **(a)** tasa de reconocimiento obtenida con diferentes cantidades de bloques de píxeles contiguos. **(b)** igual que (a), pero solo para las imágenes que tienen oclusión con anteojos negros. **(b)** igual que (a), pero solo para las imágenes que tienen oclusión con bufandas.

puede observarse que el aumento de la cantidad de bloques no afecta tanto a las imágenes ocluidas con anteojos negros. Este fenómeno se debe a que las imágenes que tienen anteojos negros tienen una menor proporción de píxeles ocluidos que las que tienen bufandas y por ende, la probabilidad de que los bloques de píxeles no contengan píxeles ocluidos es menor.

Con respecto a la cantidad de píxeles de los conjuntos F_i, hay que destacar que las pruebas que tuvieron alrededor del 15 % del total de los píxeles resultaron ser más estables que aquellas que solo tenían el $6,25\,\%$.

El siguiente experimento fue fijar la cantidad de bloques de píxeles a 2 y a 1250 píxeles en los conjuntos F_i ($15,5\,\%$ del total de la imagen) y luego variar el umbral τ.

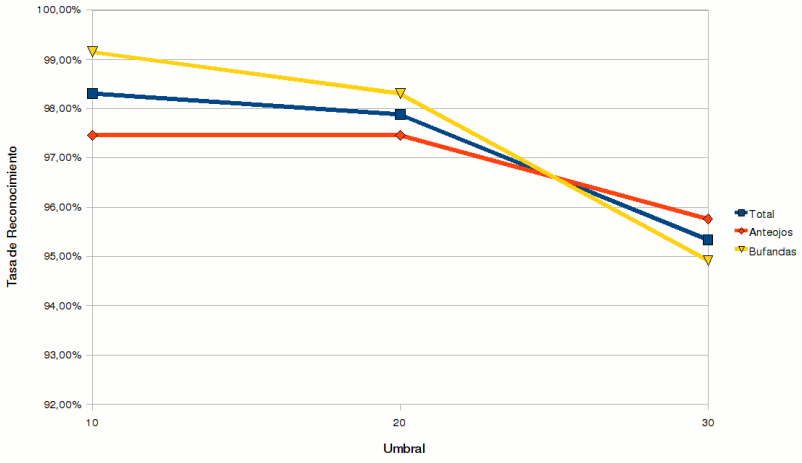


Figura 4.18: Tasa de reconocimiento obtenida utilizando el método de detectar la oclusión, para luego excluir dicha porción. En estas pruebas se utilizaron dos bloques de píxeles contiguos, con el $15,5\,\%$ de los píxeles en los conjuntos F_i y se varió el umbral τ.

La figura 4.18 muestra los resultados de variar el umbral τ. Experimentalmente, obtuvimos los mejores resultados utilizando $\tau = 10$.

En la figura 4.19 incluimos algunos ejemplos de rostros ocluidos y como el algoritmo detectó dicha zona ocluida, para luego reconocer satisfactoriamente al sujeto utilizando el resto de la imagen.

Si observamos detenidamente las imágenes de la figura 4.19 que muestran las zonas de oclusión detectadas, vemos que algunas de ellas tienen pequeñas regiones no ocluidas las cuales fueron detectadas como oclusión y viceversa. Dado que el algoritmo de reconocimiento es robusto a pequeñas zonas oclui-

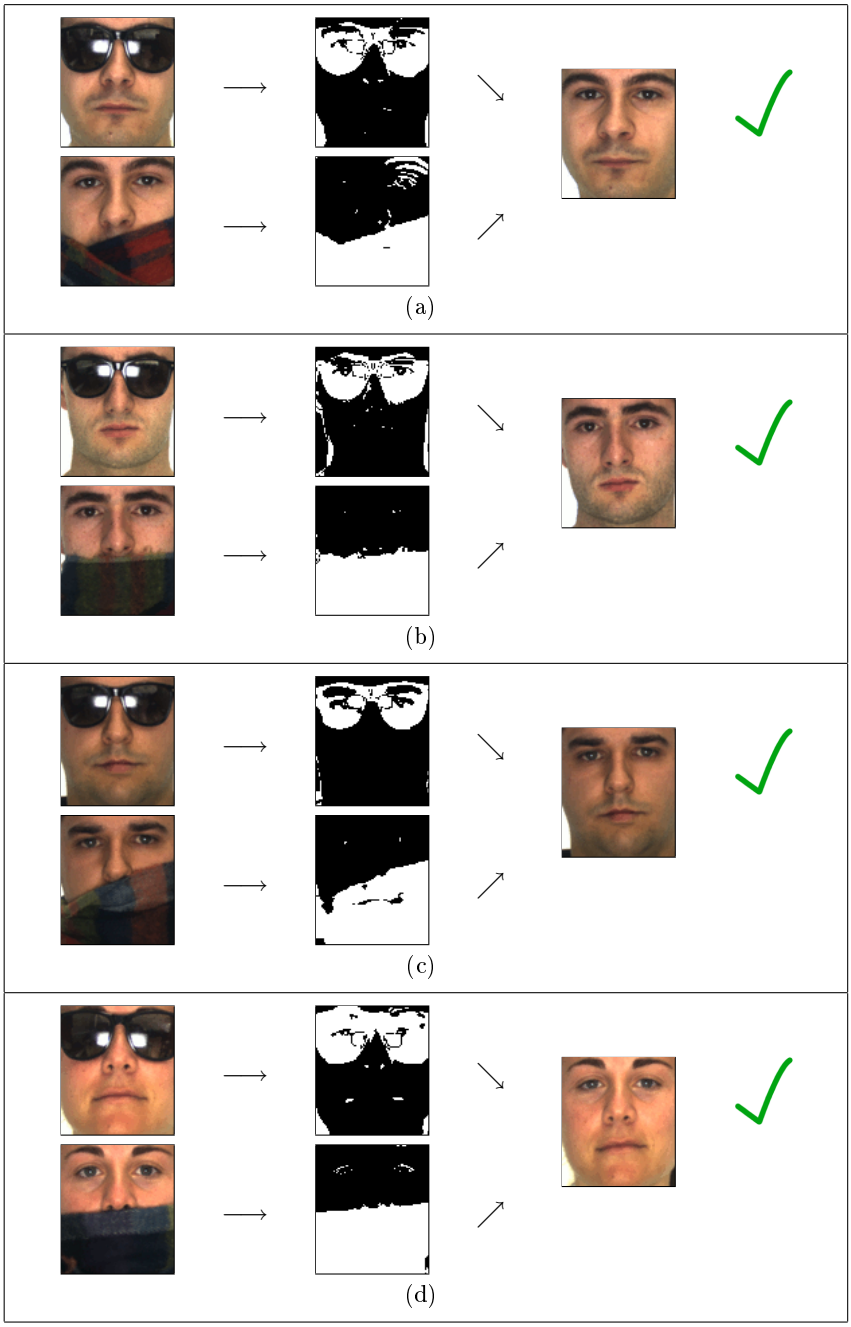


Figura 4.19: Reconocimiento de imágenes con oclusión correspondientes a cuatro individuos, (a), (b), (c) y (d). En esta figura se muestran dos imágenes con oclusión por individuo. Luego se muestra la zona ocluida detectada para cada una de estas imágenes y por último se pueden ver las imágenes sin oclusión de dichos individuos, los cuales fueron correctamente detectados por el algoritmo.

das, estos pequeños errores cometidos no afectan a la tasa de reconocimiento. Esto último se confirma con la elevada tasa de reconocimiento obtenida.

Por último, en la tabla 4.4 podemos observar comparativamente las tasas de reconocimiento más altas obtenidas con cada uno de los tres métodos de reconocimiento de rostros en presencia de oclusión.

Algoritmo	**Anteojos**	**Bufandas**	**Total**
Modelo Simple	$69,49\,\%$	$12,71\,\%$	$41,10\,\%$
Modelo Particionado	$86,44\,\%$	$95,76\,\%$	$91,10\,\%$
Detección de la oclusión	$\mathbf{97,46}\,\%$	$\mathbf{99,15}\,\%$	$\mathbf{98,31}\,\%$

Tabla 4.4: Tasas de reconocimiento obtenidas con cada uno de los métodos de reconocimiento de rostros en presencia de oclusión.

En la siguiente sección comentaremos los resultados obtenidos combinado el método de detección de rostros y el de reconocimiento.

4.4. Pruebas de integradoras

En esta sección comentamos los resultados obtenidos combinando el método de detección de rostros con el de reconocimiento. Buscamos entonces reconocer individuos en imágenes de caras encontradas por el detector y luego centradas y recortadas en forma automática, sin ningún tipo de intervención manual. Es importante destacar, que este procedimiento se realiza tanto con las imágenes usadas para test, como con aquellas utilizadas en el diccionario de caras del algoritmo de reconocimiento.

El detector fue entrenado utilizando los parámetros que, en base a las mediciones efectuadas en la sección 4.2, dieron mejores resultados. Es decir, que el detector elegido consta de un árbol de clasificadores con una ramificación y en donde cada uno de los clasificadores fuertes que lo componen, utiliza a su vez árboles *CART* con dos clasificadores simples.

Para estas pruebas usamos las mismas bases de datos utilizadas en la sección de reconocimiento (ver 4.3), es decir las bases de datos de caras *Extended YALE B* y *AR Database*.

La base de datos *Extended YALE B* consta de imágenes correspondientes a 38 sujetos y cada uno de ellos posee múltiples imágenes obtenidas bajo diferentes condiciones de iluminación. Los creadores de esta base de datos,

clasificaron las imágenes en función de la iluminación en cuatro conjuntos a los que denominan: *Set 1*, *Set 2*, *Set 3* y *Set 4*. El primero de ellos contiene las imágenes en donde la iluminación es frontal y/o difiere de esta en un ángulo menor a 12 grados. El segundo tiene imágenes en donde la iluminación está corrida entre 20 y 25 grados, el tercero entre 35 y 50 grados y el último, el cual posee las imágenes con iluminación más extrema, entre 60 y 77 grados.

La tabla 4.5 muestra la tasa de detección y la tasa de falsos positivos

Imágenes	**Tasa de detección**	**Tasa de falsos positivos**
Set 1	$97,30\,\%$	$0,00\,\%$ (0)
Set 2	$96,85\,\%$	$0,68\,\%$ (3)
Set 3	$92,57\,\%$	$0,90\,\%$ (4)
Set 4	$72,78\,\%$	$0,39\,\%$ (2)
Total	$\mathbf{88,29}\,\%$	$\mathbf{0,54}\,\%$ (**9**)

Tabla 4.5: Tasa de detección y de falsos positivos obtenidos en la base de datos *Extended YALE B*

obtenidos en la etapa de detección. Como puede apreciarse, la tasa de detección se reduce significativamente en las imágenes que tienen la iluminación más extrema (*Set 4*).

Luego se armó el diccionario de caras utilizando las caras detectadas correspondientes a las imágenes que fueron utilizadas en el diccionario de caras de las pruebas de la sección 4.3.1.1. Esto se hizo para poder comparar cuánto afecta la tasa de reconocimiento el hecho de que las imágenes no fueron alineadas manualmente. Con ese diccionario de caras y usando los valores $\epsilon = 0{,}05$ y $d = 256$ [1] se aplicó el algoritmo de reconocimiento a las imágenes de caras detectadas restantes.

La tasa de reconocimiento obtenida con este procedimiento fue del $\mathbf{93,89}\,\%$. Recordemos que con las imágenes de esta base de datos de caras que fueron centradas y recortadas manualmente el reconocimiento era efectivo en un $100\,\%$. Esta diferencia en la tasa de reconocimiento se debe a que unas pocas caras detectadas no estaban bien centradas y otras fueron detectadas en una escala diferente al resto de las imágenes.

Con todo esto, podemos decir que la tasa de efectividad del sistema obtenida con la base de datos de caras *Extended YALE B* es de $88,29\,\% \times 93,89\,\% = \mathbf{82,90}\,\%$.

[1] Los valores de los parámetros ϵ y d se corresponden con los que proporcionaron los mejores resultados para esta base de datos de caras en la sección 4.3.1.1

Luego procedimos a repetir los experimentos pero utilizando la base de datos de caras *AR Database*. En esta base de datos, utilizamos las imágenes de los mismos 60 individuos seleccionados para las pruebas de las secciones 4.3.1.2 y 4.3.2.

<table>
<tr><th colspan="2">Tipo de imágenes</th><th colspan="2">Tasa de detección obtenida</th><th></th></tr>
<tr><td colspan="2">Sin oclusión</td><td colspan="2">100,00 %</td><td rowspan="3">90,22 %</td></tr>
<tr><td rowspan="2">Con oclusión</td><td>Anteojos</td><td>80,79 %</td><td rowspan="2">78,81 %</td></tr>
<tr><td>Bufandas</td><td>76,84 %</td></tr>
</table>

Tabla 4.6: Tasa de detección obtenida en la base de datos *AR Database*

Los resultados de la detección pueden apreciarse en la tabla 4.6. Como puede observarse, las imágenes que no se encuentran ocluidas fueron detectadas en su totalidad. Sin embargo, esto no sucede cuando las imágenes se encuentran parcialmente ocluidas.

<table>
<tr><th colspan="2">Tipo de imágenes</th><th colspan="2">Tasa de falsos positivos</th><th></th></tr>
<tr><td colspan="2">Sin oclusión</td><td colspan="2">0,49 % (4)</td><td rowspan="3">1,44 % (22)</td></tr>
<tr><td rowspan="2">Con oclusión</td><td>Anteojos</td><td>4,80 % (17)</td><td rowspan="2">2,45 % (18)</td></tr>
<tr><td>Bufandas</td><td>0,28 % (1)</td></tr>
</table>

Tabla 4.7: Tasa de falsos positivos obtenida en la detección utilizando la base de datos *AR Database*. Entre paréntesis se indica la cantidad de falsos positivos obtenida.

Por otro lado, la tabla 4.7 muestra la cantidad de falsos positivos detectados. Hay que aclarar que una pequeña cantidad de falsos positivos en la etapa de detección no genera mayores inconvenientes ya que éstos son rechazados luego en la etapa de reconocimiento debido a que el coeficiente SCI resultante suele ser un orden de magnitud inferior respecto cuando la imagen a reconocer es un rostro.

Luego se armó el diccionario de caras utilizando el mismo procedimiento empleado con la base de caras *Extended YALE B*, pero para las imágenes sin oclusión. Es decir que el diccionario se armó con las caras detectadas correspondientes a las imágenes que fueron utilizadas en el diccionario de caras de las pruebas de la sección 4.3.1.2. Con ese diccionario de caras y usando los

valores $\epsilon = 0{,}05$ y $d = 256$ [2] se aplicó el algoritmo de reconocimiento a las imágenes de caras detectadas restantes.

La tasa de reconocimiento obtenida para las imágenes sin oclusión fue de un $\mathbf{87,96}\,\%$, es decir, alrededor de un $7\,\%$ menos que utilizando imágenes centradas y recortadas manualmente (ver sección 4.3.1.2). Por otro lado, los falsos positivos tuvieron un SCI menor a $0,0001$ y al ser un valor tan chico, estos fueron descartados.

Podemos entonces decir que la tasa de efectividad del sistema obtenida con las imágenes sin oclusión base de datos de caras *AR Database* es de $100,00\,\% \times 87,96\,\% = \mathbf{87,96}\,\%$.

Luego intentamos hacer la misma prueba pero con las imágenes con oclusión. Pero, a diferencia de las sin oclusión, la mayoría (más del $70\,\%$) de las imágenes se encuentran mal centradas y en diferentes escalas. Esto produce que el reconocimiento no sea factible en dichas imágenes. Al ver estos resultados, llegamos a la conclusión que se necesita una etapa entre las de detección y reconocimiento que se encargue de centrar y escalar las imágenes detectadas previo al reconocimiento. Mas aún, esto es de vital importancia si las imágenes pueden contener cierto grado de oclusión. Queda entonces planteado un nuevo tema de investigación acerca de como alinear imágenes de caras que puedan o no estar parcialmente ocluidos.

En resumen, en esta sección mostramos que los métodos de detección y reconocimiento pueden utilizarse en forma conjunta y automática cuando las caras de las imágenes no se encuentran ocluidas. Se hicieron mediciones en dos bases de datos de caras y en ambas, la tasa de reconocimiento fue alrededor de un $7\,\%$ inferior a la obtenida utilizando imágenes centradas y recortadas manualmente. También se comentó que cuando las imágenes se encuentran parcialmente ocluidas, la alineación y la escala de las imágenes de caras resultantes del algoritmo de detección son muy variables, lo que impide el reconocimiento y en consecuencia es necesario agregar algún tipo de alineación entre ambas etapas.

[2] Los valores de los parámetros ϵ y d se corresponden con los que proporcionaron los mejores resultados para esta base de datos de caras en la sección 4.3.1.2

Capítulo 5

Conclusiones

En este trabajo estudiamos la detección y el reconocimiento de caras. Abordamos la detección basándonos en un método propuesto por Viola y Jones [46] el cual consiste en entrenar una cascada de clasificadores y donde cada uno de ellos está compuesto por *features* simples y que pueden ser calculados muy velozmente. Este método consta de dos etapas, la primera en donde se entrena el detector utilizando un algoritmo de *boosting*. Luego tiene una segunda etapa en donde se utiliza la cascada de clasificadores obtenida en la etapa anterior para la detección. La gran ventaja de este método es que la segunda etapa, a diferencia de la primera, puede ejecutarse muy velozmente, lo que permite detectar rostros en tiempo real aún en dispositivos con capacidad de procesamiento limitada (como ser, una cámara de fotos o un celular).

Posteriormete comentamos sucesivas mejoras que se le hicieron al algoritmo original del Viola y Jones. Estas son:

- Extender el conjunto de *features* utilizado e incorporar *features* inclinados a 45 grados.
- Agregar una etapa de post-optimización al clasificador entrenado, lo que permite reducir la tasa de falsos positivos de un clasificador.
- Utilizar GentleBoost como algoritmo de boosting en vez de AdaBoost.
- Utilizar árboles *Classification and Regression Trees* (CART) en vez de *stumps* como clasificadores débiles.
- Construir en forma automática un árbol de clasificadores fuertes en vez de una cascada.

Este método de detección de caras mostró tener excelentes resultados cuando las caras no se encuentran ocluídas. Aunque, cuando ciertas porciones del rostro se encuentran ocluídas, la tasa de detección suele caer un poco.

Luego abordamos el problema del reconocimiento de rostros mediante un novedoso método que tiene su fundamento en recientes avances en el estudio del procesamiento de señales, mas específicamente en el área de *compressed sensing*.

En la etapa de reconocimiento formamos un diccionario de caras, con las imágenes de los rostros de los sujetos conocidos por el sistema. Luego, dada una imagen a reconocer, buscamos la combinación lineal más esparsa de imágenes presentes en el diccionario que formen dicha imagen. Dado que este cálculo es NP-Hard, se aproximó dicho cálculo utilizando un sistema de ecuaciones lineales, en las condiciones planteadas en la teoría de *compressed sensing*.

Posteriormente extendimos el planteo anterior para que el reconocimiento pueda efectuarse aún cuando la imagen del rostro del sujeto a reconocer se encuentre parcialmente ocluida. Este planteo funciona bien cuando la porción ocluida del rostro es pequeña, pero comienza a fallar a medida que esta crece. Para solucionar esto, se propuso subdividir el problema del reconocimiento en pequeños problemas de reconocer fracciones del rostro. Vimos también que este segundo planteo no es lo suficientemente flexible. Es por ello que en este trabajo propusimos un nuevo método que permite detectar la zona ocluida de la imagen, para luego excluirla del reconocimiento.

Este método para detectar la oclusión consiste en tomar pequeñas porciones de la imagen y utilizando el diccionario de caras conocidas, generar una cara similar a la que se quiere detectar. Dado que la imágenes del diccionario no se encuentran ocluidas, la imagen generada se asemejará a un rostro no ocluido. Luego, por diferencia de la imagen a testear con al generada, se obtiene la región ocluida.

Tanto el método de reconocimiento sin oclusión, como el de detección de la oclusión y posterior reconocimiento, mostraron excelentes resultados.

Luego probamos ambos métodos juntos, para detectar y reconocer rostros sin ningún tipo de intervención manual. Los resultados obtenidos fueron muy buenos cuando las imágenes no estaban ocluidas. Sin embargo, también se mostró que cuando las imágenes tienen cierta porción ocluida, la etapa de detección suele devolver imágenes no alineadas y en diferentes escalas, lo que impide reconocerlas. Debido a esto, queda pendiente como futuro trabajo de investigación crear un método que permita centrar las imágenes de caras detectadas, aunque estas se encuentren parcialmente ocluidas, previo al reconocimiento.

Bibliografía

[1] Edoardo Amaldi and Viggo Kann. On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoretical Computer Science*, 209(1–2):237–260, 1998.

[2] Ronen Basri and David W. Jacobs. Lambertian reflectance and linear subspaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(2):218–233, February 2003.

[3] Peter N. Belhumeur, Joao Hespanha, and David J. Kriegman. Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720, 1997.

[4] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)*, pages 245–250, August 2001.

[5] V. Blanz, S. Romdhani, and T. Vetter. Face Identification across Different Poses and Illuminations with a 3D Morphable Model. *Automatic Face and Gesture Recognition, IEEE International Conference on*, 0:0202+, 2002.

[6] Volker Blanz, Kristina Scherbaum, and Hans-Peter Seidel. Fitting a Morphable Model to 3D Scans of Faces. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, 2007.

[7] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3D faces. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[8] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.

[9] E. J. Candes. Compressive sampling. In *Proceedings of the International Congress of Mathematicians, Madrid, Spain*, volume 3, pages 1433–1452, 2006.

[10] E. J. Candes and M. B. Wakin. An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21–30, March 2008.

[11] Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, August 2006.

[12] Emmanuel J. Candès and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE Transactions on Information Theory*, 52(12):5406–5425, 2006.

[13] Scott Shaobing Chen, David L. Donoho, Michael, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20:33–61, 1998.

[14] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:681–685, 2001.

[15] David L. Donoho. For most large underdetermined systems of linear equations the minimal ℓ1-norm solution is also the sparsest solution. *Comm. Pure Appl. Math*, 59:797–829, 2004.

[16] G. J. Edwards, T. F. Cootes, and C. J. Taylor. Face recognition using active appearance models. pages 581+. 1998.

[17] Kamran Etemad and Rama Chellappa. Discriminant analysis for recognition of human face images. *Journal of Optical Society of America A*, 14:1724–1733, 1997.

[18] Xiaoli Zhang Fern and Carla E. Brodley. Random projection for high dimensional data clustering: A cluster ensemble approach. In Tom Fawcett and Nina Mishra, editors, *ICML*, pages 186–193. AAAI Press, 2003.

[19] Yoav Freund. Boosting a Weak Learning Algorithm by Majority. In *COLT '90: Proceedings of the third annual workshop on Computational learning theory*, pages 202–216, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[20] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm, 1996.

[21] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[22] J. Friedman, T. Hastie, and R. Tibshirani. Special invited paper. additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–374, 2000.

[23] A.S. Georghiades, P.N. Belhumeur, and D.J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 23(6):643–660, 2001.

[24] Xiaofei He, Shuicheng Yan, Yuxiao Hu, and Partha Niyogi. Face recognition using laplacianfaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(3):328–340, 2005.

[25] A. Hyvärinen. Survey on independent component analysis. *Neural Computing Surveys*, 2:94–128, 1999.

[26] Fatih Kahraman, Binnur Kurt, and Muhittin Gokmen. Robust face alignment for illumination and pose invariant face recognition. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–7, 2007.

[27] S. Kaski. Dimensionality reduction by random mapping: fast similarity computation for clustering. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 1, pages 413–418 vol.1, 1998.

[28] Kuang-Chih Lee, Jeffrey Ho, and David J. Kriegman. Acquiring linear subspaces for face recognition under variable lighting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(5):684–698, 2005.

[29] R. Lienhart, Luhong Liang, and A. Kuranov. A detector tree of boosted classifiers for real-time object detection and tracking. In *ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo*, pages 277–280, Washington, DC, USA, 2003. IEEE Computer Society.

[30] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. pages 297–304. 2003.

[31] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *IEEE ICIP 2002*, pages 900–903, 2002.

[32] Dahua Lin and Xiaoou Tang. Quality-driven face occlusion detection and recovery. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–7, 2007.

[33] J. B. Macqueen. Some methods for classification and analysis of multivariate observations. In *Procedings of the Fifth Berkeley Symposium on Math, Statistics, and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[34] John Maindonald. Modern multivariate statistical techniques: Regression, classification and manifold learning. *Journal of Statistical Software, Book Reviews*, 29(11):1–3, 2 2009.

[35] A.M. Martinez and R. Benavente. The AR face database. Technical Report 24, CVC, 1998.

[36] Constantine Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. In *ICCV*, pages 555–562, 1998.

[37] Sam T. Roweis and Lawrence K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500):2323–2326, December 2000.

[38] M. Savvides, R. Abiantun, J. Heo, S. Park, C. Xie, and B.V.K. Vijayakumar. Partial & holistic face recognition on frgc-ii data using support vector machine. *Computer Vision and Pattern Recognition Workshop*, 0:48, 2006.

[39] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227–227, June 1990.

[40] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10(5):1299–1319, July 1998.

[41] P. Sinha, B. Balas, Y. Ostrovsky, and R. Russell. Face recognition by humans: Nineteen results all computer vision researchers should know about. *Proceedings of the IEEE*, 94(11):1948–1962, January 2007.

[42] L. Sirovich and M. Kirby. Low-dimensional procedure for the characterization of human faces. *J. Opt. Soc. Am. A*, 4(3):519–524, 1987.

[43] M. B. Stegmann, B. K. Ersbøll, and R. Larsen. FAME - a flexible appearance modelling environment. *IEEE Transactions on Medical Imaging*, 22(10):1319–1331, 2003.

[44] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[45] I. Ulusoy and C. M. Bishop. Generative versus discriminative methods for object recognition. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 258–265 vol. 2, 2005.

[46] Paul Viola and Michael Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, 2004.

[47] L. Wiskott, J. M. Fellous, N. Kuiger, and C. von der Malsburg. Face recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):775–779, July 1997.

[48] John Wright, Allen Y. Yang, Arvind Ganesh, Shankar S. Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(2):210–227, 2009.

[49] Allan Y. Yang, John Wright, Yi Ma, and S. Shankar Sastry. Feature selection in face recognition: A sparse representation perspective. Technical Report UCB/EECS-2007-99, EECS Department, University of California, Berkeley, Aug 2007.

[50] Zihan Zhou, Andrew Wagner, Hossein Mobahi, John Wright, and Yi Ma. Face recognition with contiguous occlusion using markov random fields. In *International Conference on Computer Vision (ICCV)*, 2009.