Universidad de las Ciencias Informáticas

Facultad 1

Título: Desarrollo del módulo de reconocimiento de rostros para el Motor de Categorización Inteligente de Contenido

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autores: Harold Riverol Echemendia
Yoana Pita Lorenzo

Tutor: Ing. Kiuver Kaddiel Ibañez Castro

La Habana. 13 de junio de 2012
"Año 54 de la Revolución"

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Autor:	Autor:
Harold Riverol Echemendia	Yoana Pita Lorenzo

Tutor:

Ing. Kiuver Kaddiel Ibañez Castro

Agradecimientos

Harold Riverol Echemendia

Agradezco a todo aquel que de una forma u otra me ha ayudado ser mejor persona. En especial a mis padres, a mis abuelos; a mi novia Yoana, por ser una compañera inseparable durante toda la universidad, incluso en la tesis. A mis suegros Marlen y Emilio, que han sido unos segundos padres para mí. A Kiuver, nuestro tutor por compartir su tiempo y conocimiento. A los compañeros del proyecto. A la gente del grupo 10105, los que están y los que ya no. Al piquete de Blender, algún día vamos a hacer algo grande. A la familia de la televisión, es una lástima que no hayamos podido intercambiar por más tiempo. A todos, muchas gracias.

Yoana Pita Lorenzo

Quiero agradecer a todos los que me guiaron durante mis años de aprendizaje, sobre todo a mi mamá y a mi papá, gracias por toda su dedicación, amor y sacrificio; me han demostrado que para alcanzar las metas solo hay que luchar por ellas, los amo. A la China, mi primera maestra, que con su dulzura y dedicación me enseñó a dar mis primeros pasos en el estudio, siempre será un ejemplo para mí. A mis viejas amistades, con las que compartí muchos momentos de alegría, gracias por eso; significa mucho para mí. A mi compañero, amigo y novio Harold, por estar a mi lado en los momentos de felicidad y tristeza en estos cinco años de carrera, siempre serás mi mejor recuerdo de la universidad, eres una persona muy especial para mí. A mis compañeros del grupo 10105 y al equipo de Blender, siempre los recordaré y espero no perder el contacto. A nuestro tutor por su maravillosa ayuda. A los compañeros del proyecto. A la familia de la televisión universitaria, especialmente a Serguei, Jorge y Yassany, por transmitirnos sus conocimientos, he aprendido mucho con ustedes. A toda mi familia, en especial a mi primo Ernestico, que ha sido un ejemplo y aunque no nos vemos muy a menudo me ha brindado su apoyo. En fin a todos los mencionados y los que han formado parte de mis lindas experiencias, gracias.

Dedicatoria

Harold Riverol Echemendia

Dedico este trabajo a mis abuelos, Blanca, Dulce y Paco por darme todo lo que ha estado a su alcance. A mis padres, Blanca Rosa y Pedro Antonio, gracias a ellos estoy aquí. Por último, a mi hermano Richard, espero que le sirva de ejemplo para el día de mañana.

Yoana Pita Lorenzo

Desde que nací, me han brindado su amor, me han dedicado su vida y sacrificio, por ellos he llegado hasta aquí. Han estado siempre ahí, apoyándome en lo que he necesitado. Qué más decir, sino que son mi razón de ser, mis padres. Por eso hoy quiero dedicarles y regalarles este gran éxito, felicidades mami y papi. Soy la persona más dichosa de este mundo por tener unos padres tan maravillosos como ustedes.

Resumen

El Motor de Categorización Inteligente de Contenido (MOCIC), permite categorizar de forma automática los contenidos digitales mediante el análisis del texto y las imágenes. Posee varios subsistemas, entre los que sobresale el Categorizador Automático de Imágenes (CAI). Durante la realización de este trabajo se desarrolla un módulo de reconocimiento de rostros para MOCIC capaz de detectar e identificar el rostro de una persona en una imagen digital. Para ello se lleva cabo una valoración crítica de los métodos para realizar el reconocimiento de rostros en imágenes digitales, propuestos en una investigación previa. Se seleccionan las herramientas que permitan el desarrollo del módulo, además se diseña e implementa el mismo. Por último, se valida su correcto funcionamiento por medio de pruebas de funcionalidad, efectividad e integración. Con el módulo de reconocimiento de rostros se pretende mejorar la efectividad de la categorización de contenidos mediante el análisis de las imágenes. Como resultado se obtiene un detector de rostros con una efectividad de 81,5% y un identificador de rostros con una efectividad de 88,9%. La integración del módulo desarrollado con MOCIC se lleva a cabo, a través del intercambio de mensajes con el módulo controlador, el cual es el encargado de comunicar los restantes subsistemas de MOCIC.

Palabras clave: Categorización de contenidos, Reconocimiento de rostros, Detección de rostros, Identificación de rostros.

Índice general

Introducción

La información generada y almacenada[1] en formato digital en un docente de producción de la Universidad de las Ciencias Informáticas (UCI) fue estimada mediante la realización de una encuesta, la cual arrojó un promedio de 7.55 Gb de almacenamiento por estación de trabajo, para un total de 2416 Gb de información. Su categorización[2] por parte de una persona, teniendo en cuenta que, categorizar un archivo toma como promedio 40 segundos y en 1 Gb caben aproximadamente 193 archivos PDF, demoraría 648 días a razón de 8 horas diarias de forma ininterrumpida, tarea que no resulta viable realizar de forma manual (ver Anexo A).

El Motor de Categorización Inteligente de Contenido (MOCIC) es una herramienta desarrollada en la UCI para la categorización automática de contenidos. Utiliza técnicas de Inteligencia Artificial (IA) y posee una arquitectura modular, adaptándose con facilidad a distintos entornos de trabajo. La categorización de contenidos puede ser utilizada, para generar listas de URLs categorizadas y así actualizar las bases de datos de los sistemas de filtrado, o para auditar los datos de las estaciones de trabajo en una institución.

MOCIC cuenta con varios módulos, entre los que sobresalen: el Módulo Preprocesador, encargado de separar el texto y las imágenes de un contenido; el Categorizador Automático de Texto (CAT), el Categorizador Automático de Imágenes (CAI) y el Módulo Decisor. Este último le asigna una categoría a un contenido teniendo en cuenta el resultado retornado por el CAT y el CAI. Todo el proceso de categorización es guiado por el Módulo Controlador de MOCIC y se puede apreciar en la Figura 1.

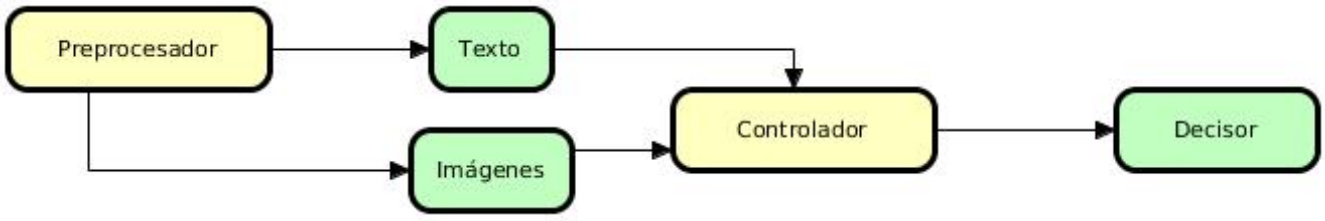
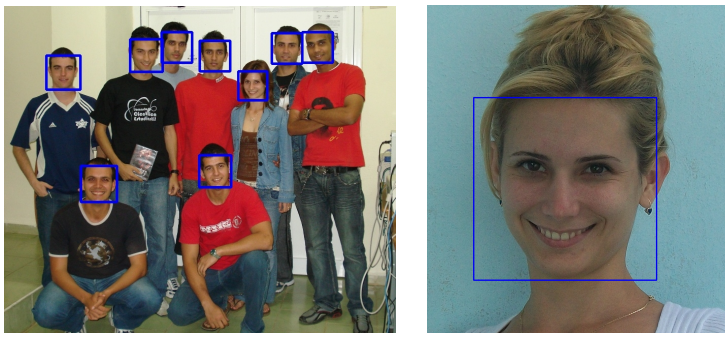


Figura 1 – Módulos de MOCIC.

[1] Se refiere solamente a libros y documentos, excluyendo fotos personales y material de ocio.

[2] Clasificación de acuerdo a su contenido.

El CAI está formado por el Módulo de Reconocimiento de Objetos (MRO) y el Módulo de Detección de Imágenes Desnudas (MDID). La detección de imágenes desnudas se realiza mediante la extracción, a las imágenes, de 7 características basadas en el color de la piel [1]. La efectividad en la detección de imágenes desnudas puede ser mejorada por medio de 2 características adicionales que necesitan de la detección de rostros. Una característica consiste en contabilizar la cantidad de rostros que tiene la imagen, como se muestra en la Figura 2 a), dado que las imágenes desnudas, por lo general contienen pocos rostros, ver Anexo B. La otra característica se relaciona con imágenes donde el rostro ocupa una gran porción de las mismas, cuya probabilidad de ocurrencia es baja en imágenes desnudas, ver Figura 2 b). Las 2 características mencionadas anteriormente tienen en común que el área de cada rostro aumenta la cantidad de piel en la imagen. Por lo que en una imagen donde hay una gran cantidad de rostros[3] o la imagen sea de un rostro[4], el detector de imágenes desnudas puede categorizarla como una imagen desnuda, generando así un Falso Positivo (FP). Un ejemplo de lo expresado anteriormente lo constituye la Figura 2 b) que fue categorizada como desnuda por el MDID y sin embargo no lo es. Esto se debe a la gran cantidad de piel, como se muestra en la Figura 3; la cual se obtuvo luego de ser aplicado el proceso de segmentación por el color de la piel a la Figura 2 b).



a) b)

Figura 2 – Detección de Rostros, a) Imagen con muchos rostros. b) El rostro ocupa gran porción de la imagen.

[3]Es muy común en sitios de aglomeración de personas, como estadios y eventos.

[4]Esto es muy común en las imágenes que son tomadas para la identificación, tal como imágenes de pasaporte y solapín.

Figura 3 – Imagen con gran cantidad de piel.

El componente encargado de asignar una categoría a un contenido en MOCIC es el Módulo Decisor (MD). El MD utiliza una serie de descriptores proporcionados por el CAT y el CAI, los cuales son empleados por algoritmos de IA para realizar el proceso de categorización. Dado que las técnicas de IA no son cien por ciento efectivas, la selección de descriptores que sean representativos es decisiva para lograr una buena efectividad.

De forma empírica se detectó que, de poder identificar una persona en una imagen digital se añadiría un descriptor más al MD, lo que podría contribuir a la efectividad de MOCIC. El descriptor pudiera ser representativo si se tiene en cuenta que en su mayoría, las personas se agrupan en diferentes categorías, entre las cuales sobresalen: deportistas, artistas, gobernantes y científicos. Por ejemplo, un contenido que posea imágenes de deportistas tiene una alta probabilidad de ser de deportes; así como un contenido donde, en sus imágenes, se reconozcan a diferentes políticos o gobernantes, tiene gran probabilidad de ser gubernamental.

MOCIC puede ser implantado para controlar el acceso en ambientes cerrados, en sistemas de video-vigilancia, para la identificación automática de personas, como en la aduana y para la obtención de estadísticas de participación de personas a determinados locales. Estas aplicaciones prácticas tienen en común que necesitan de un mecanismo capaz de realizar el reconocimiento facial. El proceso de reconocimiento facial engloba la detección y la identificación de rostros. La detección consiste en localizar, en una imagen digital, la región que ocupa un rostro humano; y la identificación es el mecanismo encargado de determinar a quién pertenece el rostro detectado.

Debido a la necesidad en MOCIC de un componente que permita reconocer el rostro de una persona dentro de una imagen, surge el siguiente problema a resolver: ¿Cómo reconocer, en una imagen digital,

el rostro de una persona, y así contribuir a la efectividad de MOCIC?

El objeto de estudio de esta investigación lo constituye el proceso de reconocimiento de rostros. El objetivo general que se persigue con este trabajo de diploma es: desarrollar un módulo que le permita a MOCIC reconocer los rostros en una imagen digital para contribuir a la efectividad de la categorización de contenidos. Los objetivos específicos derivados del objetivo general son:

- Caracterizar, a partir del estudio del arte, aplicaciones y herramientas que realicen el reconocimiento de rostros.
- Implementar el módulo que le permita a MOCIC reconocer los rostros en una imagen digital.
- Validar el módulo implementado teniendo en cuenta su funcionamiento.

El campo de acción está constituido por: los métodos y técnicas para el desarrollo del módulo de reconocimiento de rostros para MOCIC.

Métodos científicos empleados.

Métodos teóricos.

- Analítico-Sintético: Se utilizó para realizar el estudio teórico y en la investigación acerca del proceso de reconocimiento de rostros en una imagen digital.
- Análisis Histórico-Lógico: Se aplicó para seleccionar el método de reconocimiento de rostros a utilizar, partiendo del análisis de los antecedentes existentes.

Métodos empíricos.

- La encuesta: Se utilizó para obtener datos estadísticos acerca de la información almacenada por algunos de los usuarios de la UCI respecto a la cantidad de documentos existentes en sus estaciones de trabajo.

Estructura del contenido

El documento estará estructurado por 3 capítulos, los cuales se describen a continuación:

El primer capítulo consta fundamentalmente de la actualización internacional y nacional de las soluciones existentes para el reconocimiento de rostros. Se describen las principales etapas del proceso de reconocimiento de rostros y finalmente se realiza un estudio de las tecnologías y herramientas, así como la metodología para el desarrollo de la aplicación.

En el segundo capítulo se realiza la propuesta del sistema y se explican los algoritmos seleccionados para el reconocimiento de rostros. Se establecen los requisitos funcionales y no funcionales. Se describen las principales historias de usuarios asociadas a los requisitos funcionales, así como la arquitectura del sistema y los patrones de diseño. Finalmente, se presenta el diagrama de clases y la descripción asociada a las principales clases.

En el tercer capítulo se muestran algunos de los artefactos relacionados con la implementación del sistema, como los diagramas de componentes y despliegue. Se valida el sistema implementado mediante pruebas de funcionalidad, efectividad e integración.

Reglas definidas:

- Las palabras en otro idioma se escribirán con letra *cursiva*.
- Las palabras escritas completamente en mayúsculas, representarán acrónimos y estarán explicadas en capítulo nombrado Acrónimos.
- Los fragmentos de código se escribirán con la familia `monospace`.

Capítulo 1

Acercamiento al reconocimiento de rostros.

Introducción

El reconocimiento de rostros depende de la efectividad que se logre en la detección y posteriormente en la identificación de rostros. Por tal motivo en el presente capítulo se realizará un estudio del estado del arte y un análisis de los métodos, herramientas y algoritmos adecuados para la realización del sistema.

1.1. Soluciones para el reconocimiento de rostros

Antes de comenzar el desarrollo del sistema se hace necesario realizar un estudio de algunas soluciones que utilizan sistemas biométricos que incluyan el reconocimiento facial. Este estudio resulta útil para identificar algún componente que pudiera ser reutilizado, así como para conocer en qué dirección se mueve el reconocimiento de rostros.

1.1.1. Internacional

En el campo del reconocimiento facial existen compañías que brindan soluciones integrales a la medida, ofrecen servicios y desarrollan aplicaciones utilizando estas técnicas. A continuación se ofrecen ejemplos de algunas de ellas.

- Avalon Biometrics: Compañía privada fundada en noviembre del 2001 en el Reino Unido e incorporada en 2004 en Madrid, España. Ofrece soluciones hechas a la medida, servicios de consultoría y administración de proyectos para el sector público. Brinda asesoría a organismos gubernamentales

y contratistas, en el diseño, la evaluación e implementación de sistemas de documentos de identificación así como su operación por medio de un proceso seguro. Incluye aplicaciones de verificación e identificación rostro y huella dactilar, documento de viaje y verificación de visa, antecedentes penales, reconocimiento de matrículas, sistemas móviles de inspección; para integrar un completo Sistema de Administración Fronterizo a escala nacional, integrado en todos los Cruces Fronterizos (Aire, Tierra y Mar) [2].

- Smarti: Herramienta que combina el control de acceso biométrico basado en el reconocimiento facial y de voz. Posee módulos de control de acceso, tiempo y asistencia, así como la automatización de hogares y edificios, además de teléfonos e intercomunicadores IP. Impide el uso no autorizado de tarjetas de acceso, códigos PIN y llaves. Incorpora además en sus soluciones, cámaras infrarrojas de Circuito Cerrado de Televisión (CCTV) para monitorizar en tiempo real y registrar todos los sucesos, con el fin de detectar cualquier alteración de los demás sistemas. La independencia de protocolo le permite conectar casi cualquier dispositivo electrónico al sistema, pudiendo operar solo o conectado con redes de computadoras [3].

- PittPatt: Pittsburgh Pattern Recognition, grupo creado en el 2004 cuyas raíces provienen de una investigación en Instituto de Robótica de la Universidad Carnegie Mellon en la década de los 90, el cual ha sido adquirido recientemente por Google. En Google, la tecnología de visión por computadora está ya en el corazón de muchos productos existentes[1] [4].

- Airborne Biometrics Group (ABG): Fundado con el objetivo de ofrecer una solución integral en el campo de reconocimiento facial. Desarrolla el producto FaceFirst el cual posee una arquitectura en la nube[2] y es capaz de alertar cuando una persona de interés es identificada [5].

- Face.com: Compañía israelí especializada en el desarrollo de aplicaciones de reconocimiento facial que funcionan eficientemente a escala Web, algunas de sus aplicaciones son utilizadas en Facebook y Twitter, además poseen una API libre en fase beta para el desarrollo de aplicaciones [6].

  Ejemplos de aplicaciones:

  Photo Finder: Busca las fotos personales publicadas en la red sugiriendo etiquetar las que no se encuentran etiquetadas. Es la aplicación disponible de mayor precisión.

  Photo Tagger: Permite seccionar álbumes de fotos, separa las personas en grupos para luego

[1] Como la Búsqueda de Imagen, YouTube, Picasa y Gafas.

[2] Es un paradigma que permite ofrecer servicios de computación a través de Internet.

sugerir su etiquetado de una sola vez. Photo Tagger permite etiquetar un álbum de fotos en una fracción del tiempo que normalmente demoraría este proceso.

Celebrityfindr: Busca fotos de celebridades en Twitter para realizar un seguimiento por medio de software de reconocimiento facial.

- PAM Face Authentication: Aplicación diseñada para permitir la autenticación facial por medio de una *webcam* en sistemas operativos GNU/Linux. Originalmente desarrollada como parte de Google Summer of Code (GSoC) en 2008 y liberada bajo la licencia General Public License (GPL) versión 3 [7].

Las soluciones mencionadas anteriormente, tienen probada efectividad y calidad. Pero a excepción de PAM Face Authentication, son software privativo y no se conocen con exactitud los algoritmos utilizados para el reconocimiento facial, por lo que no se consideran una opción. Además, su uso implica altos costos por concepto de licencias y gran dependencia de los propietarios.

La tecnología de Face.com aunque posee una API libre, tiene como desventajas las siguientes: está diseñada para ser utilizada en línea, está sujeta a limitaciones de uso y se encuentra en fase beta de desarrollo, por lo que no es estable.

PAM Face Authentication tiene la desventaja de ser liberada bajo la licencia GPLv3, la cual tiene carácter viral, lo que obliga a que cualquier aplicación derivada debe ser liberada bajo la misma licencia. Y la política del centro no contempla liberar el Motor de Categorización Inteligente de Contenido (MOCIC) bajo la licencia GPL.

1.1.2. Nacional

Se tiene referencia de investigaciones realizadas en nuestro país y experiencia acumulada en el reconocimiento facial. En la Universidad de las Ciencias Informáticas (UCI), el Departamento de Biometría del Centro de Identificación y Seguridad Digital (CISED) y el Departamento de Tecnologías perteneciente al Centro de Informatización de Entidades (CEIGE), tienen líneas de investigación acerca de este tema. Además, el Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV)[3] cuenta con experiencia en el campo de reconocimiento de patrones y biometría. Como resultado de [8], se obtuvo, en el centro

[3] http://www.cenatav.co.cu/es/

CEIGE, un sistema de autenticación mediante reconocimiento facial utilizando los algoritmos AdaBoost y PCA; implementados en la biblioteca OpenCV. Estas investigaciones se han tenido en cuenta durante el desarrollo del presente trabajo de diploma.

1.2. Principales etapas del proceso de reconocimiento de rostros

Existe, en la bibliografía relacionada con el tema, diversidad de criterios acerca de las etapas del proceso de reconocimiento de rostros. Tanto es así que algunos autores asumen que los rostros ya han sido detectados, sin embargo, en el presente trabajo se considera que la detección de rostros es un eslabón importante dentro de este proceso. Por lo que se adoptan, al igual que en [9, p. 18], las etapas del reconocimiento de rostros:

- Detección del rostro.
- Representación del rostro o extracción de características.
- Clasificación del rostro.

La representación y la clasificación, a su vez conforman la identificación. Para la realización del presente trabajo de diploma, se toman como punto de partida las técnicas y algoritmos para el reconocimiento de rostros, propuestos en [10]; investigación llevada a cabo como parte del ciclo de desarrollo del proyecto. A continuación se realiza una valoración crítica a los algoritmos propuestos, con el objetivo de validar su correcto funcionamiento y así ser utilizados para el desarrollo del módulo de reconocimiento de rostros de MOCIC.

1.2.1. Detección del rostro

Constituye uno de los primeros pasos en los sistemas de reconocimiento facial, de cuya calidad dependen las siguientes etapas. Consiste en localizar los rostros existentes en el área que ocupa una imagen digital y es influenciado, en gran medida, por las condiciones de iluminación bajo las cuales fue tomada la imagen y la calidad de la misma. Hay que tener en cuenta otros elementos que pueden tornar engorrosa la detección y se relacionan con el rostro en si, como la orientación, el vello facial, la expresión facial y el uso de accesorios, entre los cuales sobresalen las gafas y sombreros [9, 11].

Los métodos de detección de rostros pueden dividirse en 2 grupos [10]:

- Métodos basados en reglas. Los cuales toman en cuenta los principales elementos que forman un rostro, tales como ojos, nariz, boca y establecen relaciones entre ellos, como por ejemplo la distancia entre los ojos.
- Métodos estadísticos. Operan sin asumir información previa de la tipología de un rostro, se basan en extraer las principales características que diferencian una imagen de un rostro a una imagen que no es un rostro. A este grupo pertenece el algoritmo de detección de rostros AdaBoost.

El algoritmo AdaBoost, propuesto para la detección de rostros en [10], esta diseñado para detectar cualquier objeto y posee además una buena efectividad. Reduce 15 veces el tiempo necesario para la detección de objetos en general [12, p. 152], logrando realizar este proceso en tiempo real. El algoritmo clasificador consiste en un árbol de decisión compuesto por una serie de clasificadores débiles, los cuales están dispuestos en forma de cascada formando una sucesión, lo cual trae como resultado un clasificador fuerte con alto grado de efectividad. Luego del estudio de [10, 8, 12, 13], se considera que AdaBoost, el cual fue validado mediante un prototipo funcional, puede ser aplicado en la etapa de detección de rostros.

1.2.2. Representación del rostro o extracción de características

La representación consiste en compactar los datos de la imagen con el objetivo de obtener una mínima cantidad de descriptores o características útiles para la clasificación, y así desechar información no relevante [14]. Está estrechamente vinculada con la clasificación y según las características que se quieren extraer, así será la representación. Algunas representaciones utilizadas son [9, p. 20]:

- Imágenes como matrices bidimensionales en escala de grises, para disminuir el tiempo de cómputo. Los métodos de reducción de vectores Principal Component Analisis (PCA)[4] y Linear Discriminant Analisis (LDA)[5].
- Vectores de características.

El algoritmo propuesto en [10] para la representación es el PCA, el cual logra reducir significativamente la dimensión de los datos; del orden de la cantidad de píxeles de la imagen, al orden de la cantidad de imágenes de entrenamiento. Por ejemplo, si se tiene una imagen de $N \times N$ píxeles, $N = 100$ se obtiene un

[4] Conocido también como *Eigenfaces*.

[5] Conocido también como *Fisherfaces*.

vector de 10000 componentes[6]; y una colección de entrenamiento de M imágenes, $M = 100$ se obtiene un vector de 99 componentes. PCA reduce la dimensión de los datos de $N \times N$ a $M - 1$, donde $M << N \times N$ $(100 << 10000)$ logrando así un tiempo de procesamiento relativamente bajo [15, p. 96]. Ofrece buenos resultados cuando las imágenes de entrenamiento y las imágenes de prueba son similares, aunque hay que señalar que disminuye su efectividad si existen grandes cambios de iluminación u orientación en las imágenes. Por último se puede mencionar que PCA requiere que todas las imágenes, tanto de entrenamiento como de prueba, posean un mismo tamaño.

El estudio de bibliografía actualizada en el campo del reconocimiento de patrones en imágenes, permite concluir que existen algoritmos que poseen gran relevancia en el reconocimiento de rostros, entre los cuales sobresalen Independent Component Analisis (ICA), LDA y Elastic Bunch Graph Matching (EBGM), además de novedosas técnicas de reconocimiento en 3D. Aunque se considera que el uso del algoritmo PCA para la representación y extracción de características constituye una buena elección para la realización de una primera versión del módulo de reconocimiento de rostros de MOCIC. Dado que se encuentra implementado en la biblioteca de procesamiento de imágenes OpenCV y sus debilidades pueden ser mitigadas mediante el preprocesamiento de las imágenes. En posteriores versiones puede ser estudiada la aplicación de alguno de los algoritmos mencionados anteriormente.

1.2.3. Clasificación del rostro

La etapa de clasificación, por lo general, es la última dentro del proceso de reconocimiento facial. Se refiere al hecho de asociar una imagen de un rostro a un nombre o persona. Está estrechamente vinculada con la representación [9], por lo que se hace necesaria la selección de un par representación-clasificación que ofrezca buenos resultados.

En [10] se propone utilizar una Red Neuronal Artificial (RNA) de tipo Radial Basis Function (RBF), la cual es una variante de red neuronal que surge como alternativa al Multi Layer Perceptron (MLP). Posee una arquitectura de tres capas, la capa de entrada, la oculta y la de salida. Tiene como característica que las neuronas de la capa oculta son activadas mediante funciones no lineales de base radial y en la capa de salida mediante una función lineal [16] como se muestra en la Figura 1.1.

[6] Se puede expresar también como un punto en un espacio de 10000 dimensiones.

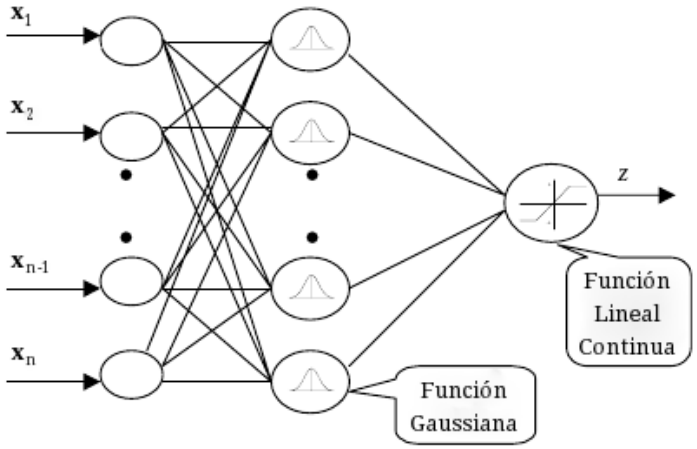


Figura 1.1 – Topología de la RNA de tipo RBF, fuente: [16].

Una función comúnmente utilizada es la campana de Gauss definida por:

$$\varphi_j(x_i) = exp\left(-\frac{\|x_i - c_j\|^2}{2{\sigma_j}^2}\right), j = 1,2,\ldots,n; i = 1,2,\ldots,N \tag{1.1}$$

donde σ es el ancho de la gaussiana, c el vector que representa el centro de la j-ésima neurona, x el vector perteneciente al patrón de entrada, n el número de neuronas de la capa oculta y N el número de patrones de entrada.

La utilización de PCA y RBFNN para la identificación en el módulo de reconocimiento de rostros de MOCIC resulta una buena elección a partir del análisis de los resultados obtenidos en [17] y la consulta de bibliografía internacional.

1.3. Tecnologías a utilizar

Elegir correctamente las herramientas, agiliza el proceso de desarrollo de software, por lo que la selección de estas se realiza según las necesidades del trabajo a realizar. En esta sección se mencionan y describen las herramientas a utilizar, las cuales deben correr sobre el sistema operativo GNU/Linux.

1.3.1. Bibliotecas

Grupo o colección de funciones, clases y algoritmos que proporciona al programador facilidades para la construcción eficiente de aplicaciones [18]. Para la realización del presente trabajo de diploma existen restricciones con la licencia, que debe ser libre y a la vez permitir su uso comercial. se investigó acerca de las bibliotecas QuickRBF y OpenCV, cuyas características se exponen a continuación.

Características de QuickRBF [19]:

- Paquete de código abierto liberado bajo la licencia BSD[7].
- Es multiplataforma y funciona en sistemas operativos como, Windows y GNU/Linux.
- Constituye una implementación eficiente de las redes neuronales artificiales de función de base radial.
- Posee funcionalidades para la normalización de los datos de entrenamiento y de clasificación, así como para la selección de los centros.

Características de OpenCV [20]:

- OpenCV es una biblioteca de código abierto, originalmente desarrollada por Intel.
- Liberada para uso comercial y de investigación bajo la licencia BSD[8].
- Es multiplataforma y funciona en sistemas operativos como, Windows y GNU/Linux.
- Enfoca principalmente hacia el procesamiento de imágenes en tiempo real.
- Está programada en los lenguajes Pyhton y C++.
- Contiene más de 500 funciones que abarcan muchas áreas en el campo de visión por computadora, incluyendo calibración de cámaras, visión estéreo y visión robótica.

Por las características mencionadas anteriormente se decide utilizarlas durante el desarrollo del trabajo.

[7] Permite el uso de código libre dentro aplicaciones no libres.

[8] Permite que sea usada libremente siempre que se cumplan las condiciones de la licencia.

1.3.2. Lenguaje de programación

Los lenguajes de programación son notaciones para describir la interacción entre las personas y las máquinas [21].

Se propone C++ para el desarrollo del trabajo de diploma debido que el CAI está implementado en este lenguaje, lo cual facilita la integración. Además, las bibliotecas OpenCV y QuickRBF están implementadas en C++ y C respectivamente.

1.3.3. Entorno de Desarrollo Integrado IDE

Conjunto de herramientas que ayudan al desarrollo de una aplicación [22]. La mayoría de los IDEs poseen herramientas para completar código y se componen de: un editor de código, un compilador, un intérprete, herramientas de automatización y un depurador[9].

En la actualidad existen IDEs de desarrollo que ofrecen la posibilidad de utilizar varios lenguajes de programación, como es el caso del NetBeans IDE[10]. En el presente trabajo de diploma, por la experiencia de los autores en su uso para el desarrollo de aplicaciones, se ha seleccionado el NetBeans IDE, pues se han obtenido resultados favorables mediante su uso en anteriores desarrollos.

1.3.4. Metodología de desarrollo

Es una versión amplia y detallada de un ciclo de vida completo de desarrollo de sistemas que incluye: reglas, procedimientos, métodos, herramientas, funciones individuales y en grupo por cada tarea, productos resultantes, así como normas de calidad [23]. La utilización de una metodología de desarrollo de software ayuda al equipo, durante todo el ciclo de vida del proyecto, a realizar las tareas necesarias en el tiempo y el orden requerido para que el producto final se obtenga con la mayor eficiencia y cumpla con las funcionalidades requeridas por el cliente.

[9] Programa usado para probar y eliminar los errores de otros programas

[10] http://netbeans.org

Metodología de desarrollo de software Scrum+XP.

La metodología de desarrollo de software SXP, es la unión de 2 metodologías ágiles, producto de una fusión llevada a cabo en la UCI. Ambas proporcionan al proyecto una eficiente gestión del trabajo en equipo y una programación rápida o extrema. A continuación se presentan las características principales de SXP [23]:

- Consiste en una programación rápida o extrema.
- Está diseñada como metodología de desarrollo para proyectos de pequeños equipos de trabajo.
- Se orienta a una entrega rápida de resultados y una alta flexibilidad.
- El usuario final forma parte del equipo de desarrollo, lo cual garantiza el éxito del producto.
- Se basa en un enfoque iterativo e incremental, donde se realizan entregas frecuentes, donde cada entrega o liberación del producto representa un incremento sobre la anterior.

Por las diferentes características mencionadas anteriormente y dado que MOCIC la emplea, se selecciona SXP, lo cual garantiza la compatibilidad de la documentación. Además, proporciona al equipo de desarrollo una eficiente guía para la realización del software y permite culminar en un tiempo prudente las diferentes fases del proyecto.

1.3.5. Lenguaje de modelado

El UML, es una notación con que se construyen sistemas por medio de conceptos orientados a objetos [24, p. 4]. Su utilización proporciona una mejor visualización de la estructura del sistema a construir y mejora la comunicación entre los miembros del equipo usando un lenguaje gráfico. Permite especificar, visualizar y construir los artefactos de los sistemas de software.

Ha sido seleccionado como lenguaje de modelado por los siguientes elementos: es un lenguaje común entre los desarrolladores del equipo, permite construir un modelo del sistema que se desea implementar y por la experiencia del equipo de desarrollo en su utilización.

1.3.6. Herramienta CASE

Las herramientas CASE comprenden varios tipos de programas utilizados para apoyar las actividades del proceso de desarrollo de software, como el análisis de requerimientos, el modelado de sistemas y las pruebas [25, p. 10]. En ocasiones son utilizadas como herramientas para asistir el proceso de realización del diseño de un software, como por ejemplo el Visual Paradigm for UML, del cual a continuación se mencionan sus principales características [26]:

- Posee un entorno de creación de diagramas para UML 2.1.
- Proporciona el uso de un lenguaje estándar común a todo el equipo de desarrollo.
- Posee capacidades de ingeniería directa[11] e inversa[12].
- Disponibilidad de múltiples versiones.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas.
- Posibilita la realización de los siguientes diagramas:
  - Clase
  - Caso de uso
  - Actividad
  - Secuencia
  - Colaboración
  - Estado
  - Despliegue
  - Componente

Por las potencialidades que ofrece esta herramienta, se decide hacer uso de la misma durante el diseño de la aplicación. Dado que ha sido utilizada para el diseño de aplicaciones anteriores obteniéndose buenos resultados, lo cual denota experiencia en el trabajo con esta herramienta.

1.3.7. Otras herramientas

Para la implementación de la aplicación se necesita el uso de las herramientas anteriormente mencionadas, las cuales agilizan el proceso de desarrollo del software, el cual debe estar acompañado del documento de la tesis de grado. Por tal motivo es importante lograr una elaboración eficiente del documento, por ello resulta necesario auxiliarse de las siguientes herramientas:

[11] Proceso de producción de código.

[12] Proceso de construir especificaciones de un mayor nivel de abstracción partiendo del código fuente.

- LaTeX: Es un sistema de composición de textos para generar documentos científicos, artículos, informes técnicos, libros y presentaciones de diapositivas de alta calidad, el cual permite la organización eficiente de la estructura del documento y la elaboración del mismo en el tiempo requerido.
- Kile: Es un IDE de desarrollo para LaTeX. Posee comando avanzados de edición, una interfaz intuitiva que posibilita el desarrollo fluido del documento.
- Gimp: Es un programa para la edición de imágenes digitales. Contiene funcionalidades principales como: la creación de figuras o fotografías y el tratamiento de imágenes en capas.

Conclusiones

Con el estudio de las principales soluciones existentes en el campo del reconocimiento facial y la valoración de los algoritmos propuestos en [10] se concluye que:

- El algoritmo AdaBoost es una buena elección para la detección de rostros debido a su rapidez y efectividad.
- El par PCA-RBFANN como forma de representación y algoritmo de clasificación respectivamente, brindan buenos resultados en la identificación de rostros.
- Las bibliotecas OpenCV y QuickRBF, dado que están liberadas bajo la licencia BSD pueden ser utilizadas para el desarrollo del sistema.

Capítulo 2

Propuesta del módulo de reconocimiento de rostros de MOCIC.

Introducción

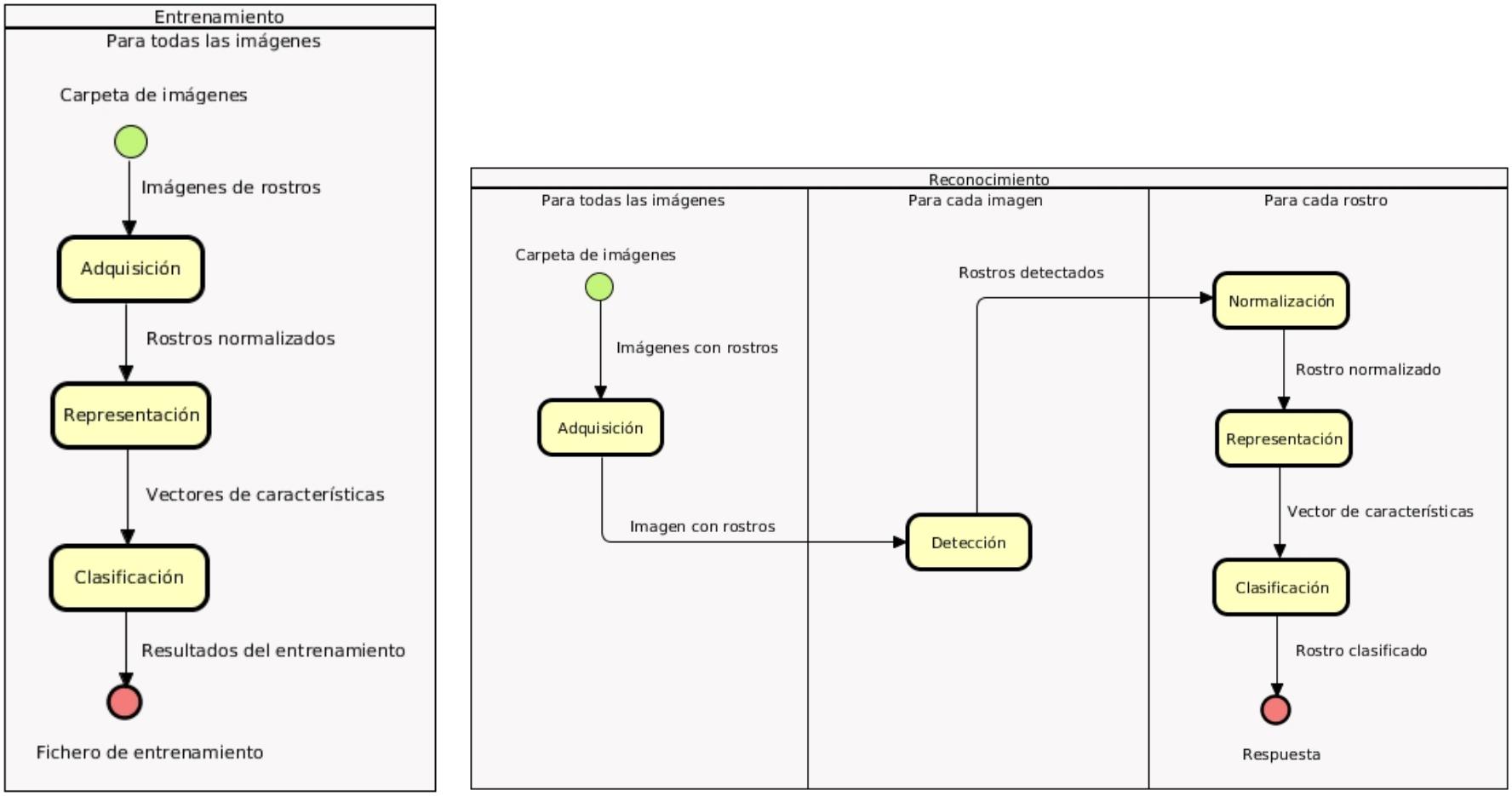
En el presente capítulo se profundiza en el estudio de los algoritmos seleccionados para el reconocimiento de rostros y se definen las principales características del sistema. Para lo cual se identifican los requisitos funcionales y no funcionales; se describe el funcionamiento del sistema y se diseña el módulo propuesto.

2.1. Propuesta de sistema

Se propone un módulo de reconocimiento de rostros capaz de detectar e identificar el rostro de una persona en una imagen digital, que además se integre a MOCIC y se comunique con el módulo decisor. El sistema contará con dos modos de trabajo bien definidos y mutuamente excluyentes: el entrenamiento[1] y el reconocimiento. Los cuales se seleccionarán por parte del usuario mediante un fichero de configuración cuya dirección se introducirá al inicio de la ejecución a través de la terminal.

Como se planteó en el capítulo anterior, el proceso de reconocimiento se divide en 3 etapas fundamentales, la detección, la representación y la clasificación. El algoritmo seleccionado para la representación es el PCA, descrito en la sección 1.2.2. El cual requiere que las imágenes de los rostros tengan el mismo tamaño y es sensible a cambios de iluminación y orientación. Por tal motivo se adiciona, antes de la representación, la etapa de normalización, cuya función es la de estandarizar las imágenes de los rostros por tamaño, orientación y color. Los procesos generales de entrenamiento y reconocimiento se pueden apreciar en la Figura 2.1 a) y 2.1 b) respectivamente, y se describen a continuación.

[1] Se refiere al entrenamiento para la identificación.



a) b)

Figura 2.1 – Procesos generales, a) Entrenamiento. b) Reconocimiento.

El entrenamiento consiste en proporcionarle al sistema, una colección de imágenes de rostros normalizados pertenecientes a las personas que se desean reconocer. Estas imágenes, en la etapa de representación, son reducidas con PCA y los vectores resultantes son utilizados para entrenar el algoritmo de clasificación. El resultado obtenido del entrenamiento se guarda en un fichero para su posterior utilización en el reconocimiento y cada vez que se adicione una nueva persona para ser identificada es necesario entrenar nuevamente. Esto se debe a que los algoritmos seleccionados para la representación y clasificación, construyen un modelo a partir de las imágenes de los rostros que se desean identificar. Por lo que, identificar el rostro de una nueva persona requiere proporcionar, a los algoritmos, un conjunto de imágenes que se unen a las que ya existen con el objetivo de construir un nuevo modelo.

Para el reconocimiento se le proporciona una carpeta con imágenes, para cada imagen, el sistema detecta los rostros. Luego, cada rostro es normalizado, reducido mediante PCA y clasificado. Obteniéndose como respuesta del sistema si el rostro es conocido o desconocido, en caso de ser conocido, a qué persona pertenece. Dado que las personas pueden agruparse en categorías, en caso que se requiera, el sistema es capaz de devolver a que categoría pertenece una imagen teniendo en cuenta a las personas identificadas en ella.

2.1.1. Algoritmo para la detección

El algoritmo AdaBoost utiliza una ventana que recorre la imagen comprobando en cada posición la existencia del objeto con el cual fue entrenado, en este caso es un rostro. El tamaño de la ventana está dado por el tamaño mínimo de los rostros que se desean detectar y una vez finalizado el recorrido de la imagen es aumentado sucesivamente hasta cubrir el total de la imagen. Finalmente, el clasificador devuelve una secuencia de rectángulos que indican las posiciones de los rostros detectados en la imagen.

La biblioteca OpenCV cuenta con un clasificador de objetos basado en el método AdaBoost, el cual puede ser invocado mediante la función `cvHaarDetectObjects(image, cascade, storage, scale_factor, min_neighbors, flags, min_size)`, donde `image` es la imagen de entrada, `cascade` es el entrenamiento cargado a partir de un fichero XML, `scale_factor` es el factor con que aumenta el tamaño de la ventana de búsqueda, por defecto es 1.1 que representa aumento del 10 %, `min_neighbors` es el número mínimo de rectángulos vecinos que conforman un objeto y `min_size` es el tamaño mínimo de la ventana de búsqueda. En este caso, como se requiere detectar rostros humanos, se utiliza el fichero de entrenamien-

to haarcascade_frontalface_default.xml disponible como parte del código fuente de la OpenCV. Luego de ejecutada, la función retorna una secuencia de rectángulos que representan la posición y el área que ocupan los rostros detectados dentro de la imagen. A partir de esta secuencia y la imagen original, cada rostro es extraído y almacenado como una imagen para continuar a la siguiente etapa del reconocimiento.

A partir de pruebas realizadas en la sección 3.4.2, se detectó que el algoritmo seleccionado posee un 72.8% de efectividad, la cual se ve comprometida debido a la alta ocurrencia de FP. Por lo que se adicionaron dos características con el objetivo de filtrar el resultado del detector y así disminuir la cantidad de FP. La primera característica es la relación entre el área y el perímetro de la mayor región de piel en el rostro, la cual es conocida como *compactness*[2] y es un valor real entre 0 y 1. Se estableció el rango con mayor probabilidad de ser un rostro, ver sección 3.4.2, por lo que los rostros candidatos que estén fuera del mismo, son descartados. La segunda característica se relaciona con la cantidad de regiones de piel del rostro. La cantidad aproximada de regiones de piel de los rostros humanos se calculó, ver sección 3.4.2 y los rostros candidatos que se encuentran fuera del rango no pasan a la siguiente etapa. Estas características tienen en común que requieren que la imagen esté segmentada. La segmentación por el color de la piel, no es más que la separación de una imagen en una imagen binaria, donde los píxeles que tienen el color de la piel toman valor 1 y los que no, toman valor 0.

Debido a que el proceso de identificación requiere de imágenes de rostros con buena calidad se decidió descartar también los rostros candidatos que son relativamente pequeños con respecto a la imagen donde están contenidos, por lo que los rostros cuyo tamaño es menor al 5% del total de la imagen son descartados. Luego de realizar pruebas, como se puede observar en la sección 3.4, se obtuvo como resultado que con la utilización de las dos características mencionadas anteriormente se obtuvo una efectividad de 81.53%, mejorando así en un 8.71% la efectividad inicial.

2.1.2. Algoritmo para la normalización

Por tamaño:

Una vez extraídas, las imágenes de los rostros son normalizadas, como muestra la Figura 2.6, por tamaño, orientación y color antes de pasar a la etapa de representación. Con respecto al tamaño, se

[2]En la bibliografía anglosajona

normalizan a 100×100 píxeles, por lo que los rostros de menor tamaño detectados pueden perder nitidez y aportar poca información a la identificación. En este caso se valorará la posibilidad de que sean descartados.

Por orientación:

La normalización por orientación se torna un poco más difícil dado que es necesario conocer la inclinación del rostro, dato que no proviene de la etapa de detección. Para resolver el problema que representa el desconocimiento de la inclinación, se identifica que los ojos constituyen una buena característica para determinar si un rostro está inclinado o no. En caso de conocer la posición de los ojos, con operaciones sencillas de trigonometría, se puede determinar el ángulo de inclinación del rostro con respecto al eje horizontal.

La posición de cada uno de los ojos está dada por un punto formado por las coordenadas (x, y). Con esos puntos es posible calcular la pendiente (m) de la recta que los une. A su vez, dicha pendiente es igual a la tangente del ángulo A. Finalmente, utilizando la función arco-tangente se puede calcular el ángulo A que corresponde a la inclinación del rostro con respecto al eje horizontal. Una vez conocida la inclinación del rostro, se procede a realizar la rotación con el ángulo A . Este proceso se puede observar en la Figura 2.2.

Sea:

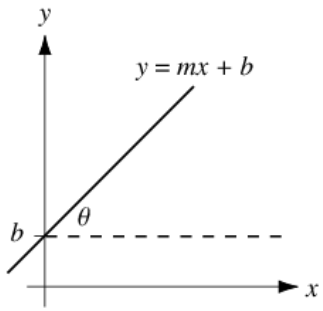
$(x_1; y_1)$: coordenada correspondiente a la posición del ojo 1.

$(x_2; y_2)$: coordenada correspondiente a la posición del ojo 2.

y

$$m = \frac{\triangle y}{\triangle x} = tan\theta$$

$$m = \frac{y_2 - y_1}{x_2 - x_1} = tan\theta$$



entonces,

$$tan\, A = m$$

por tanto:

$$A = atan(m)$$

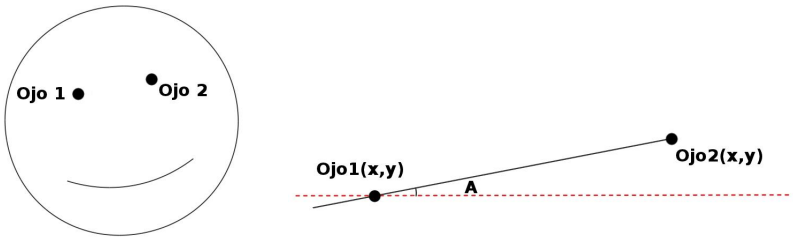


Figura 2.2 – Cálculo del ángulo de inclinación del rostro.

Con el objetivo de localizar los ojos se decide hacer uso de las potencialidades del algoritmo AdaBoost, utilizado en la etapa de detección de rostros. En este caso se utiliza el fichero de entrenamiento cascade_eye.xml para detectar ojos, disponible en el código fuente de la biblioteca OpenCV, el resultado de aplicar el algoritmo se observa en la Figura 2.3.

Figura 2.3 – Ojos Detectados, fuente: [27].

Como se observó en la sección 1.2.1, el algoritmo para la detección de los ojos trae consigo cantidades considerables de FP. Por lo que se decide realizar un filtrado de los ojos candidatos, al igual que se realizó con la detección de rostros. En este caso se tienen en cuenta dos características, el tamaño y la localización del ojo con respecto al rostro. Relacionado con el tamaño, se descartan los ojos candidatos cuyas dimensiones estén fuera del rango aproximado de los ojos con respecto al rostro. Relacionado con la localización, se pueden descartar si se tiene en cuenta que, si se divide el rostro a la mitad mediante una línea horizontal, los ojos se encuentran en la mitad superior; y si a su vez esta mitad es dividida horizontalmente en dos partes iguales lo ojos generalmente se encuentran en la parte inferior, ver Figura 2.4.

La efectividad inicial obtenida mediante la realización de las pruebas descritas en la sección 3.4.2 fue de 75%. Por otra parte, con el filtrado de los ojos candidatos se mejoró en un 7%, obteniéndose una

efectividad para la detección de ojos de 82%, lo cual demuestra que filtrar los resultados del algoritmo AdaBoost para disminuir la ocurrencia de FP es una buena opción para elevar efectividad del algoritmo.

Figura 2.4 – Localización de los ojos, fuente: [27].

Por color:

Para la normalización por color, las imágenes son convertidas a escala de grises. Mediante esta operación una imagen es representada mediante 256^3 tonalidades de gris, lo cual minimiza las variaciones de color. Pero esto no resulta suficiente si se tiene en cuenta que, en muchas ocasiones, las imágenes no son tomadas por profesionales; lo cual ocasiona significativas variaciones de iluminación.

Con el objetivo de estandarizar las condiciones de iluminación de los rostros, se utiliza la técnica de ecualización del histograma. El histograma de una imagen representa la frecuencia relativa de ocurrencia de varios niveles gris en la imagen. Matemáticamente hablando para una imagen digital con niveles de grises en el rango [0, L-1][4], el histograma es una función discreta $p(r_k) = n_k/N$ donde r_k es el k-ésimo nivel de gris y n_k es el número de píxeles en la imagen con ese nivel de gris. N representa el total de píxeles de la imagen y $k = 0, 1 \ldots L$-1. El histograma brinda una descripción global de la imagen. Por ejemplo, si el histograma es estrecho significa que la imagen es pobremente visible; en cambio si el histograma está ampliamente distribuido, la imagen tiene buen contraste y mejora su visibilidad, ver Figura 2.5 a). La forma del histograma de una imagen revela información importante acerca del contraste, la cual puede ser usada para el mejoramiento de las imágenes [28, p. 110]. Un ejemplo de lo anteriormente expuesto se muestra en la Figura 2.5 b).

[3] [0 –255]

[4] En nuestro caso $L = 256$ por lo que el rango es [0,255].

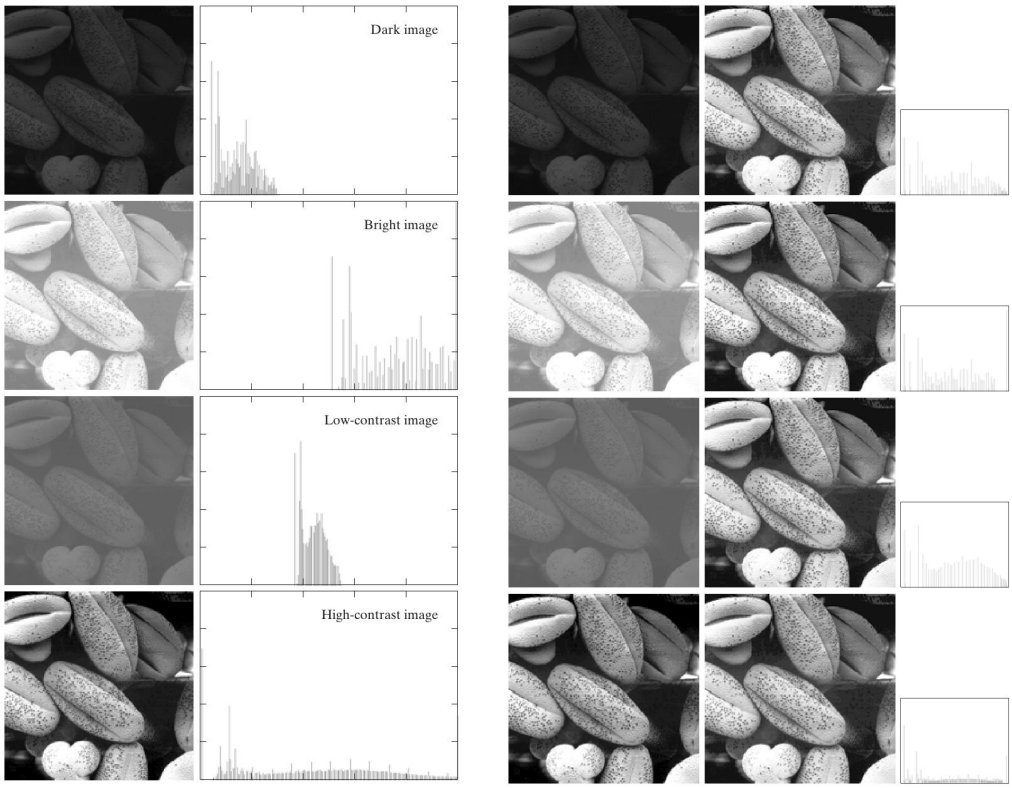


Figura 2.5 – Ecualización del histograma, a) Cuatro tipos básicos de imágenes: oscura, clara, con bajo contraste, con alto contraste, y sus correspondientes histogramas. b) Resultados de la ecualización del histograma, fuente: [14, p. 90].

La biblioteca OpenCV posee la función `cvEqualizeHist(src,dst)` encargada de ecualizar el histograma de una imagen en escala de grises, la cual resulta de mucha utilidad para la normalización de los rostros por color, donde `src` es la imagen de entrada y `dst` la imagen ecualizada.

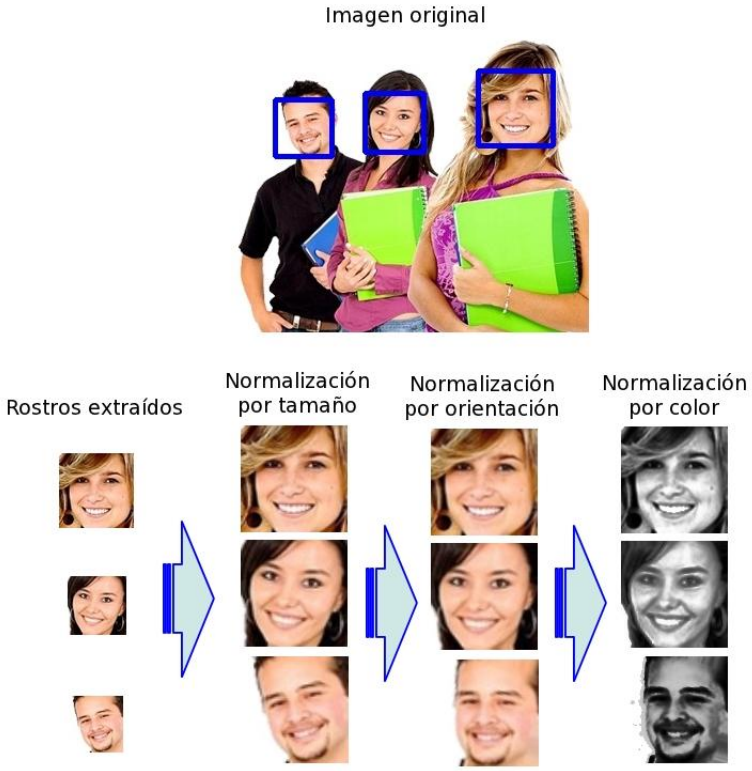


Figura 2.6 – Normalización de los rostros, fuente: [27].

2.1.3. Algoritmo para la representación

La etapa que sigue a la normalización es la representación, la cual es de gran importancia, pues de la calidad de sus resultados depende la clasificación. Se ha seleccionado el algoritmo PCA para extraer las principales características de los rostros y reducir significativamente la dimensión de los datos.

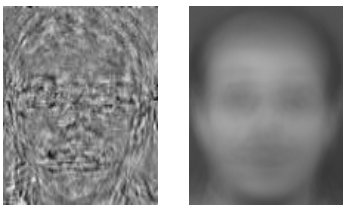
Una imagen digital puede ser expresada en forma de vector de números. Si el ancho y el alto de la imagen son w y h píxeles, el número de los componentes del vector es $w \times h$ y, donde cada componente es el valor en escala de grises de un píxel. Este vector se forma por la concatenación de las filas de píxeles de la imagen [9, p. 26]. Una imagen de un rostro de 100×100 píxeles representada por un vector de 10000 componentes, puede ser reducida mediante PCA, a la cantidad de imágenes de entrenamiento menos 1. Por lo que se pudiera asumir que con colecciones de entrenamiento relativamente grandes se comprometería la reducción de los datos. El PCA, sin embargo, tiene la ventaja de que no es necesario almacenar todas las componentes, dado que solamente son las primeras las que ofrecen información

relevante para la identificación de los rostros.

Para una mejor compresión del algoritmo es necesario señalar que un autovector (*eigenvector*) es un vector cuya dimensión es la misma que las imágenes iniciales y por lo tanto se puede ver como una imagen, en el llamado espacio imagen. Al hacerlo así, da la apariencia de imágenes de caras, pero muy diferentes del conjunto inicial. Por esta razón, en la literatura se conocen como *eigenfaces* [9, p. 30].

En la Figura 2.7 se muestran los primeros cuatro *eigenfaces* de una colección. Como se puede observar tienen el aspecto de rostros humanos, sin embargo el *eigenface* número 190, como se observa en la Figura 2.8 a) no ofrece mucha información, interpretándose prácticamente como ruido.

Figura 2.7 – Cuatro primeros *eigenfaces* de una colección de rostros.



a) b)

Figura 2.8 – Resultados del algoritmo PCA, a) *Eigenface* 190. b) Rostro Medio.

Otro resultado del algoritmo PCA es el rostro medio, como se muestra en la Figura 2.8 b), el cual no es más que el resultado de la normalización del conjunto de entrenamiento [9, p. 31]. Durante el reconocimiento, el rostro que se desea identificar se proyecta sobre el rostro medio, se extraen los componentes principales y estos últimos pasan a la etapa de clasificación.

Algunos autores señalan que los primeros 30 componentes principales aportan suficiente información para identificar un rostro. Otros sugieren que algunos de los 45 a 80, aportan más información que alguno de los primeros 15. Teniendo en cuenta la diversidad de criterios existentes, la cantidad de componentes a tener en cuenta para la etapa de representación se determinó mediante la realización de pruebas, ver

sección 3.4.2, las cuales arrojaron los mejores resultados con 90 componentes principales.

2.1.4. Algoritmo para la clasificación

La clasificación se encarga de retornar si un rostro es desconocido o pertenece a alguna persona del entrenamiento. Se ha seleccionado una red neuronal artificial de función de base radial como clasificador automático.

Una red neuronal de función de base radial, como se explicó en la sección 1.2.2, posee 3 capas, la de entrada, la oculta y la de salida. Específicamente para la identificación de rostros, la cantidad de neuronas de entrada(n) corresponde al número de *eigenfaces*. La selección de la cantidad de neuronas de la capa oculta(c) para el reconocimiento facial, es un tema donde existe diversidad de criterios de varios autores: en [19] se sugiere utilizar, para colecciones menores que 10000 elementos, los datos de entrenamiento como neuronas. Por otro lado, en [17] optan por agrupar imágenes similares para seleccionar las neuronas de la capa oculta. En este trabajo se decide agrupar las imágenes de una misma persona, para tener una neurona por cada persona a identificar, y disminuir así el tiempo de procesamiento. Finalmente, el número de neuronas de la capa de salida(m) es la cantidad de personas a identificar más 1.

Por ejemplo, si se desean identificar 10 personas, se utilizan los primeros 30 *eigenfaces* y se cuenta con una colección de entrenamiento de 10 imágenes de cada persona para un total de 100 imágenes; la topología de la red neuronal quedaría como muestra la Figura 2.9, con $n = 30$, $c = 10$ y $m = 11$.

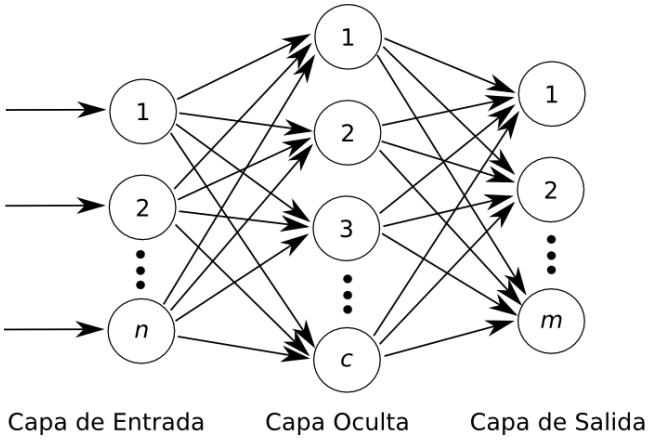


Figura 2.9 – Topología de la red neuronal.

La topología de la red neuronal propuesta para el módulo de reconocimiento de rostros de MOCIC es la siguiente:

$n = 90$

$c =$ cantidad de personas a identificar.

$m =$ cantidad de personas a identificar + 1.

Es importante además definir el ancho de la función gaussiana[5], su selección se realizó mediante las pruebas descritas en la sección 3.4.2. Los mejores resultados se obtuvieron con σ=3 para el entrenamiento y σ=2.25 para la identificación. Con el objetivo de determinar si un rostro es conocido o desconocido es necesario establecer un umbral de aceptación, por lo que un rostro que se encuentre por debajo del umbral será clasificado como desconocido. Para establecer un adecuado umbral de aceptación se requirió de la realización de pruebas, las cuales se detallan en la sección 3.4 y los mejores resultados se obtuvieron con un umbral de 0.95.

Para la construcción del clasificador se hace uso de la biblioteca QuickRBF, la cual es un paquete con una implementación de una red neuronal de tipo RBF. QuickRBF ofrece algunas herramientas que complementan el trabajo con la red neuronal, entre las que se encuentra una para normalizar los datos de entrada. No obstante, para su uso fue necesario realizar algunas modificaciones en su código fuente para adaptarla a las necesidades del sistema. Entre las modificaciones se encuentran las funcionalidades de guardar los datos del entrenamiento en un fichero y cargarlos en memoria durante la clasificación. Así como la función `principalQuickRBF(center, trainfile, sigma, confidence, train, nFaces, nEigen, numPerson, vectorData)`, que puede ser usada tanto para el entrenamiento como para la clasificación, lo cual hace más sencilla su utilización.

2.2. Requisitos funcionales (RF)

Especifican los servicios que debe cumplir el sistema, la forma en que debe reaccionar a entradas particulares y de cómo se debe comportar ante situaciones específicas, y en algunos casos pueden declarar lo que el sistema no debe hacer [25].

RF1. Detectar rostros en una imagen.

[5]Función de activación de las neuronas de la capa oculta.

RF2. Entrenar el identificador de rostros.

RF2.1. Salvar el entrenamiento.

RF3. Identificar rostros en una imagen.

RF3.1. Cargar el entrenamiento.

RF4. Integrar a MOCIC.

RF5. Salvar las trazas del módulo en un fichero de texto.

RF6. Cargar fichero de configuración con todos los parámetros de la aplicación.

2.3. Requisitos no funcionales (RNF)

Son las restricciones de los servicios o funciones ofrecidas por el sistema, incluyen restricciones de tiempo sobre el proceso de desarrollo y estándares [25].

Requisitos de usabilidad.

- Debe ser intuitivo, requiriendo un mínimo de esfuerzo por parte del usuario.

Requisitos de rendimiento.

- Debe mantener un rendimiento estable a lo largo del tiempo, utilizando un mínimo de recursos.
- Debe reconocer un rostro en un tiempo relativamente rápido.

Requisitos de portabilidad.

- Debe contar con un `.deb` que permita la instalación de forma sencilla para adaptarse a diferentes ambientes de trabajo.

Requisitos de entorno.

- Para ejecutar el software se requiere como mínimo una PC Pentium 4 con 1Gb de RAM y Sistema Operativo Ubuntu 12.04 o una versión LTS posterior.
- Para compilar el software se requiere de la bibliotecas: libcv2.1, libcv-dev, libcvaux2.1, libcvaux-dev, libhighgui2.1, libhighgui-dev, libmagic1, libmagic-dev, libfann2, libfann-dev, libmocic-common, libmocic-common-dev, libquickrbf.

- Para ejecutar el software se requiere de la bibliotecas: libcv2.1, libcvaux2.1, libhighgui2.1, libmagic1, libfann2, libmocic-common, libquickrbf.

2.4. Historias de Usuario

Una HU es una parte de una funcionalidad que es de valor para el cliente. Ofrece una manera sencilla de determinar lo que el sistema tiene que hacer, de manera que pueda ser entregado por partes [29]. Es utilizada en las metodologías ágiles tales como SXP y permite estimar el tiempo total del proyecto. A continuación se describen las principales HU.

2.4.1. Descripción de la HU: Detectar rostros en una imagen

Historia de Usuario	
Número: IMG-11	**Nombre Historia de Usuario:** Detectar rostros en una imagen.
Modificación de Historia de Usuario Número: Ninguna.	
Usuario: Harold Riverol Echemendia.	**Iteración Asignada:** Sprint 1.
Prioridad en Negocio: Muy Alta.	**Puntos Estimados:** 5
Riesgo en Desarrollo: Medio.	**Puntos Reales:** 5
Descripción: Dada una imagen, esta funcionalidad debe ser capaz de devolver la posición de los rostros contenidos en la misma. Además, debe brindar la posibilidad de imprimir en consola alguna información de interés, como por ejemplo la cantidad de rostros detectados en la imagen, así como salvar la imagen con los rostros señalados por un rectángulo que indique su posición y tamaño.	
Observaciones: La imagen debe poseer buenas condiciones de iluminación y la resolución debe ser mayor de 100×100 píxeles, para obtener resultados satisfactorios.	
Prototipo de interfaz: -	

2.4.2. Descripción de la HU: Identificar rostros en una imagen

Historia de Usuario	
Número: IMG-23	**Nombre Historia de Usuario:** Identificar rostros en una imagen.
Modificación de Historia de Usuario Número: Ninguna.	
Usuario: Harold Riverol Echemendia.	**Iteración Asignada:** Sprint 2.
Prioridad en Negocio: Muy Alta.	**Puntos Estimados:** 3
Riesgo en Desarrollo: Alta.	**Puntos Reales:** 3
Descripción: Dada una imagen de un rostro, esta funcionalidad debe identificar a que persona pertenece, retornando -1 si no identificó el rostro y si fue identificado el número de la persona a la cual pertenece. Debe imprimir el resultado en la consola.	
Observaciones: La imagen que se utilice debe poseer buenas condiciones de iluminación y la resolución debe ser mayor de 100×100 píxeles, para obtener resultados satisfactorios. Además, para poder identificar una persona, esta debe estar en el entrenamiento.	
Prototipo de interfaz: -	

2.4.3. Descripción de la HU: Integrar a MOCIC

Historia de Usuario	
Número: IMG-27	**Nombre Historia de Usuario:** Integrar a MOCIC.
Modificación de Historia de Usuario Número: Ninguna.	
Usuario: Harold Riverol Echemendia.	**Iteración Asignada:** Sprint 2.
Prioridad en Negocio: Media.	**Puntos Estimados:** 2
Riesgo en Desarrollo: Medio.	**Puntos Reales:** 2
Descripción: El sistema debe poder integrarse con MOCIC y ser capaz de comunicarse con el módulo controlador mediante el intercambio de mensajes.	
Observaciones: -	
Prototipo de interfaz: -	

2.5. Arquitectura del sistema

El sistema utilizará una arquitectura en forma de *plugins*, donde cada funcionalidad se encuentre independiente, para así permitir su activación o desactivación sin necesidad de compilar nuevamente. Los *plugins* se cargarán dinámicamente, lo cual dota al módulo de mayor flexibilidad y escalabilidad. Por ejemplo, si en un entorno de despliegue solamente se necesita detectar rostros, se activan solamente las funcionalidades relacionadas con la detección garantizando un mejor rendimiento. Se decide adoptar esta arquitectura pues resulta eficiente para manipular flujos de datos, además permite el desarrollo de sistemas flexibles. Cabe destacar que existe, en el proyecto, experiencia y conocimientos acerca de la misma. La Figura 2.10 a) muestra una vista de la arquitectura del sistema y se observa que al núcleo del sistema, llamado common se adicionan los diferentes *plugins* a utilizar. El sistema también se integra al módulo controlador de MOCIC y al Depósito, con la particularidad que pueden existir varias instancias simultáneas para el reconocimiento de rostros, como se puede observar en la Figura 2.10 b).

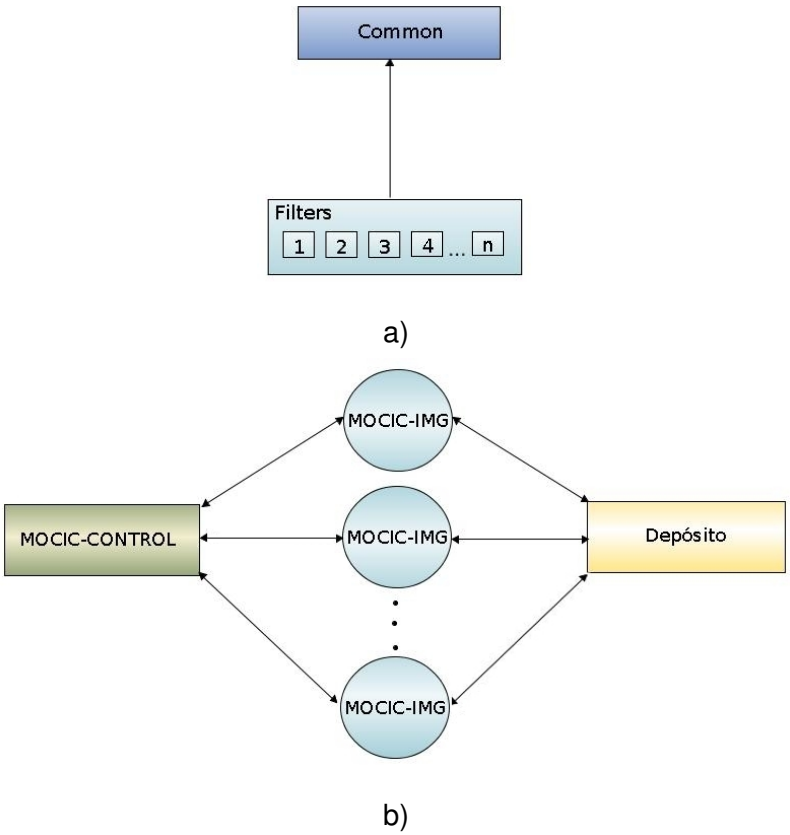


Figura 2.10 – Arquitectura, a) Del sistema. b) Varias instancias simultáneas para el módulo de reconocimiento de rostros.

2.6. Patrones de Diseño y Arquitectura utilizados

La utilización de patrones de diseño durante el desarrollo del sistema resulta de vital importancia, puesto que proveen soluciones estándares para problemas comunes de programación, basados en la experiencia acumulada a lo largo de los años por los programadores. Su adecuado empleo constituye una poderosa arma para lograr aplicaciones robustas y flexibles. Para el diseño de la solución al problema planteado en el presente trabajo de diploma de utilizaron los patrones que se relacionan a continuación.

2.6.1. Patrones Orientados a la Arquitectura de Software

Los patrones arquitectónicos expresan los esquemas estructurales fundamentales de los sistemas de software. Proveen una serie de subsistemas predefinidos, especifican las responsabilidades e incluyen reglas y directrices para la organización de las relaciones entre ellos. Representan los patrones de más alto nivel en el sistema de patrones y ayudan a especificar la estructura fundamental de una aplicación [30, pp. 25-26]. La arquitectura del sistema propuesto se enmarca dentro de las arquitecturas de flujo de datos, aplicando el patrón arquitectónico Pipes and Filters. Su selección está dada por el hecho que facilita la reutilización de componentes, además es apropiada para sistemas de procesamiento de datos que requieren pasos sucesivos.

Pipes and Filters: Se caracteriza por dividir la tarea de un sistema en varios pasos secuenciales de procesamiento llamados filtros. Los filtros están conectados por el flujo de datos a través del sistema, de manera tal que la salida de uno corresponde a la entrada del siguiente. La entrada del sistema corresponde a una fuente de datos como un fichero de texto. La salida fluye a un sumidero, como por ejemplo, un fichero o una terminal. Los datos de entrada, los filtros y el sumidero se encuentran conectados secuencialmente por tuberías [30, p. 55].

Para el sistema propuesto, la entrada corresponde a una secuencia de imágenes, los filtros son las clases encargadas de realizar diferentes etapas del proceso de reconocimiento de rostros y la salida es la clasificación de los rostros detectados.

2.6.2. Patrones GOF

Dentro de los patrones Gang of Four (GOF) se hace uso de los patrones de creación, los cuales son una guía de cómo crear objetos cuando su creación requiere tomar decisiones. Estas decisiones normalmente serán resueltas dinámicamente decidiendo que clases instanciar o sobre qué objetos se delegarán responsabilidades [31].

- Singleton: Garantiza que una clase tenga una única instancia, proporcionando un punto de acceso global a la misma. Se pone de manifiesto en la creación de las instancias de todos los filtros, entre los cuales sobresalen FaceDetect, NormalizeFace, PCA y RBFRNA. Los filtros se instancian una sola vez y todas las funcionalidades se realizan por parte de esa única instancia.
- Prototype: Es un patrón de creación a nivel de objetos. Permite especificar los tipos de objetos que se desea crear usando una instancia prototípica. Se puede usar cuando las clases a instanciar se especifican en tiempo real. Ofrece mayor flexibilidad al sistema, pues se pueden adicionar funcionalidades dinámicamente en tiempo de ejecución. Es usado para la creación de los filtros, dado que estos se cargan a partir de un fichero de configuración, lo cual permite utilizar solamente las necesarias en dependencia de la tarea que se quiere realizar. La aplicación de este patrón permite la activación de las funcionalidades sin necesidad de compilar nuevamente la aplicación.

2.6.3. GRASP: Patrones para Asignar Responsabilidades

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones [24].

- Experto: Asigna una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. Se utiliza en los filtros, entre ellos se puede mencionar FaceDetect, que tiene la responsabilidad de detectar los rostros en una imagen; PCA, responsabilizado de la reducción de vectores de las imágenes de los rostros y RBFRNA encargado de realizar la clasificación.

  Se utiliza porque conserva el encapsulamiento, pues los objetos se valen de su propia información para realizar la tarea encomendada. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. Además, el comportamiento se distribuye

entre las clases que cuentan con la información requerida, haciéndolas más fáciles de comprender y de mantener, garantizando una alta cohesión.

- Bajo Acoplamiento: El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases. Acoplamiento bajo significa que una clase no depende de muchas clases [24]. Por lo anteriormente expresado se decide hacer uso del mismo en el diseño del sistema para asignar responsabilidades que garanticen un bajo acoplamiento. Los filtros, entre los cuales se encuentran FaceDetect, NormalizeFace, PCA y RBFRNA poseen bajo acoplamiento realizando sus funciones independientemente una de la otra, lo cual garantiza que un cambio en alguna de ellas no implique un cambio en el resto del sistema.

2.7. Diagrama de clases

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases del software y de las interfaces en una aplicación [24]. La Figura 2.11 muestra la composición de las clases del módulo de reconocimiento de rostros, en el centro se encuentra el paquete common, donde sobresale la clase Filter, la cual implementa las interfaces IFilter e IFactory y de ella heredan todas las funcionalidades o filtros. Los filtros de color verde son los más importantes, por lo cual se describen a continuación, mientras que las demás clases se describen en el Anexo C.

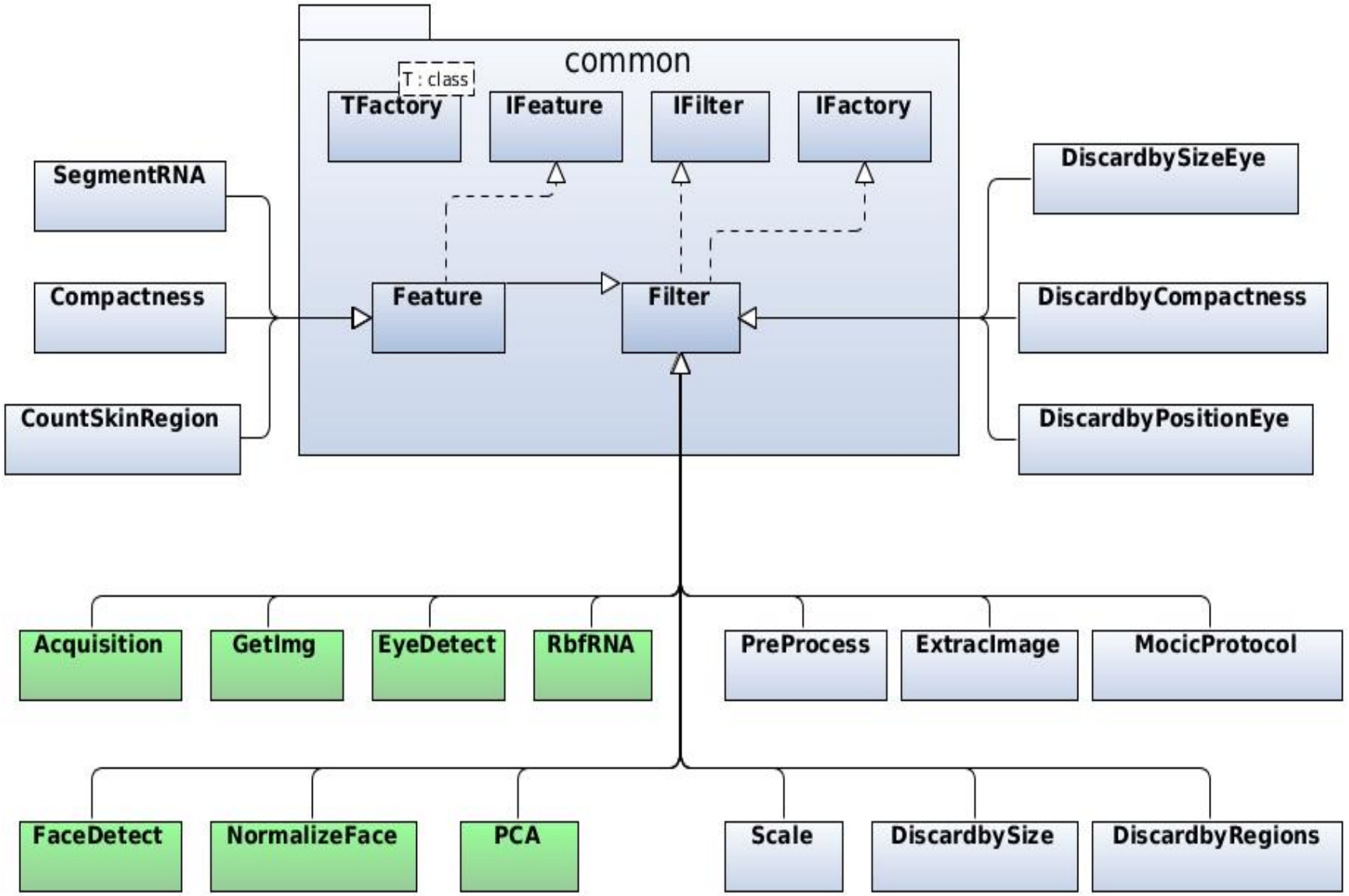


Figura 2.11 – Diagrama de clases del diseño.

2.7.1. Descripción de las clases

Nombre de la Clase: Acquisition	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`file`	`ListString*`
`delete_image`	`bool`
`train`	`bool`
Para cada responsabilidad	
Nombre:	`Acquisition()`
Descripción:	Constructor de la clase, devuelve un objeto Acquisition.
Nombre:	`~Acquisition()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Carga todas las rutas de las imágenes y se las entrega una a una a GetImg. En modo de entrenamiento devuelve una lista con todas las imágenes, y en modo de reconocimiento devuelve la categoría a la cual pertenecen las imágenes.

Tabla 2.4 – Descripción de la clase Acquisition.

Nombre de la Clase: GetImg	
Tipo de Clase: Controladora	
Atributo	**Tipo**
No posee atributos propios, solamente los de la clase Filter, de la cual hereda.	
Continúa en la próxima página	

Para cada responsabilidad	
Nombre:	`GetImg()`
Descripción:	Constructor de la clase, devuelve un objeto GetImg.
Nombre:	`~GetImg()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Recibe la ruta de la ubicación de una imagen, carga la imagen en memoria y la devuelve en forma de objeto.

Tabla 2.5 – Descripción de la clase GetImg.

Nombre de la Clase: FaceDetect	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`CvHaarClassifierCascade*`	`cascade`
`draw_rectangle`	`bool`
Para cada responsabilidad	
Nombre:	`FaceDetect()`
Descripción:	Constructor de la clase, devuelve un objeto FaceDetect.
Nombre:	`~FaceDetect()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`

Continúa en la próxima página

Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Recibe una imagen y devuelve una secuencia con los rostros detectados en la imagen.
Nombre:	`CvSeq* detect(IplImage* img, CvMemStorage* storage, int scale = 1)`
Descripción:	Detecta los rostros en una imagen mediante el algoritmo AdaBoost.
Nombre:	void draw(`IplImage* img`, `CvSeq* faces`, `int scale = 1`)
Descripción:	Dibuja un rectángulo por cada rostro detectado en la imagen señalando su localización.

Tabla 2.6 – Descripción de la clase FaceDetect.

Nombre de la Clase: EyeDetect	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`CvHaarClassifierCascade*`	`cascade`
`draw_rectangle`	`bool`
Para cada responsabilidad	
Nombre:	`EyeDetect()`
Descripción:	Constructor de la clase, devuelve un objeto EyeDetect.
Nombre:	`~EyeDetect()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.

Continúa en la próxima página

Nombre:	`Object getOutput(Object argument)`
Descripción:	Recibe una imagen de un rostro y devuelve una secuencia con los ojos detectados en la imagen.
Nombre:	`CvSeq* detect(IplImage* img, CvMemStorage* storage, int scale = 1)`
Descripción:	Detecta los ojos en una imagen mediante el algoritmo AdaBoost.
Nombre:	`void draw(IplImage* img, CvSeq* faces, int scale = 1)`
Descripción:	Dibuja un rectángulo por cada ojo detectado en la imagen del rostro señalando su localización.

Tabla 2.7 – Descripción de la clase EyeDetect.

Nombre de la Clase: NormalizeFace	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`resize`	`bool`
`equalizet`	`bool`
`gray`	`bool`
`rotate`	`bool`
`size`	`int`
Para cada responsabilidad	
Nombre:	`NormalizeFace()`
Descripción:	Constructor de la clase, devuelve un objeto NormalizeFace.
Nombre:	`~NormalizeFace()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`

Continúa en la próxima página

Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Normaliza una imagen de un rostro por tamaño, color y orientación.
Nombre:	`void rotateImg(IplImage* img_aux, IplImage* dst, CvSeq* eyes, CvRect* roi)`
Descripción:	Determina el ángulo de inclinación de una imagen de un rostro y rota la imagen en valor del ángulo de inclinación para normalizar el rostro por orientación.

Tabla 2.8 – Descripción de la clase NormalizeFace.

Nombre de la Clase: PCA	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`eigen`	`int`
`train`	`bool`
`training`	`string`
`eigenValMat`	`CvMat*`
`eigenVectArr`	`IplImage * *`
`pAvgTrainImg`	`IplImage *`
`projectedTrainFaceMat`	`CvMat*`
Para cada responsabilidad	
Nombre:	`PCA()`
Descripción:	Constructor de la clase, devuelve un objeto PCA.
Nombre:	`~PCA()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.

Continúa en la próxima página

Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Reduce los vectores de las imágenes de los rostros aplicando el método PCA implementado en la OpenCV. En modo de entrenamiento procesa todas las imágenes de entrenamiento y en modo de reconocimiento procesa un rostro a la vez.
Nombre:	`void savePersonNumberName(char *arg, int nTrainFaces, std::vector <std::string> personNames,CvMat* personNumTruthMat)`
Descripción:	Guarda en un fichero la correspondencia entre nombre de la persona a quien corresponden las imágenes de entrenamiento y el orden en que han sido cargadas dichas imágenes.
Nombre:	`void saveTrainEigenVectors(char *arg, int nTrainFaces, CvMat* personNumTruthMat)`
Descripción:	Guarda en un fichero los vectores resultantes de la aplicación del algoritmo PCA en modo de entrenamiento para ser utilizados en el entrenamiento de la red neuronal.
Nombre:	`void saveTestEigenVectors(char *arg, float *projectedTestFace, int i, CvMat* personNumTruthMat)`
Descripción:	Guarda en un fichero los vectores resultantes de la aplicación del algoritmo PCA en modo de reconocimiento para ser clasificados por la red neuronal.
Nombre:	`void saveTrainingAsXML(char *arg, int nTrainFaces, std::vector <std::string> personNames, CvMat* personNumTruthMat)`
Continúa en la próxima página	

Descripción:	Guarda en un fichero XML la información resultante de aplicar el método *PCA* en modo de entrenamiento, para ser utilizados por el propio algoritmo en modo de reconocimiento. Esta funcionalidad se usa en modo de entrenamiento.
Nombre:	`int loadTrainingXML(char *arg, CvMat * * pTrainPersonNumMat, int &nTrain, std::vector <std::string> personNames)`
Descripción:	Carga el fichero XML con la información de los rostros reducidos con PCA. Esta funcionalidad se usa en modo de reconocimiento.

Tabla 2.9 – Descripción de la clase PCA.

Nombre de la Clase: RBFRNA	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`center`	`int`
`sigma`	`int`
`confidence`	`double`
`train`	`bool`
`removefile`	`bool`
`trainin`	`string`
Para cada responsabilidad	
Nombre:	`RBFRNA()`
Descripción:	Constructor de la clase, devuelve un objeto RBFRNA.
Nombre:	`~RBFRNA()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Continúa en la próxima página	

Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Clasifica un rostro mediante una red neuronal artificial de función de base radial utilizando la librería QuickRBF modificada. Recibe un vector reducido con PCA y determina a que persona pertenece.
Nombre:	`void saveAnswerPersonTruthName(char *arg, int answer, std::string pName)`
Descripción:	Guarda en un fichero la correspondencia entre nombre de la persona a quien corresponde la imagen de prueba y el resultado retornado por la red neuronal. Es útil para calcular la efectividad de la identificación.
Nombre:	`void saveTrainEigenVectors(char *arg, int nTrainFaces, CvMat* personNumTruthMat)`
Descripción:	Guarda en un fichero los vectores resultantes de la aplicación del algoritmo PCA en modo de entrenamiento para ser utilizados en el entrenamiento de la red neuronal.
Nombre:	`void saveTestEigenVectors(char *arg, float *projectedTestFace, int i, CvMat* personNumTruthMat)`
Descripción:	Guarda en un fichero los vectores resultantes de la aplicación del algoritmo PCA en modo de reconocimiento para ser clasificados por la red neuronal.
Nombre:	`void saveTrainingAsXML(char *arg, int nTrainFaces, std::vector <std::string> personNames, CvMat* personNumTruthMat)`
Descripción:	Guarda en un fichero XML la información resultante de aplicar el método PCA en modo de entrenamiento, para ser utilizados por el propio algoritmos en modo de reconocimiento. Esta funcionalidad se usa en modo de entrenamiento.

Continúa en la próxima página

Nombre:	`int loadTrainingXML(char *arg, CvMat * * pTrainPersonNumMat, int &nTrain, std::vector <std::string> personNames)`
Descripción:	Carga el fichero XML con la información de los rostros reducidos con PCA. Esta funcionalidad se usa en modo de reconocimiento.

Tabla 2.10 – Descripción de la clase RBFRNA.

Conclusiones

Luego de definir las características que debe cumplir el Módulo de Reconocimiento de Rostros de MOCIC y de realizar el Análisis y Diseño del mismo, se concluye que:

- El sistema propuesto cumple con las funcionalidades requeridas.
- La selección de una adecuada arquitectura garantiza el desarrollo de un módulo de reconocimiento de rostros flexible.
- El uso adecuado de patrones de diseño es necesario para lograr un software con calidad.

Capítulo 3

Implementación y validación del módulo de reconocimiento de rostros para MOCIC.

Introducción

Luego de realizar el diseño de módulo propuesto y de definir su arquitectura, se procede a implementar un sistema que cumpla con los requisitos que se necesitan. En el presente capítulo se muestran los principales artefactos de la implementación del mismo. Además, se valida el módulo implementado mediante la realización de pruebas de funcionalidad, efectividad e integración.

3.1. Plan de *Releases*

En el plan de *releases* se asignan Historia de Usuario (HU) a *releases* e iteraciones. Se dividen las grandes HU en otras más pequeñas y se aplazan las menos valiosas hasta completar el tiempo disponible [29].

Release	**Descripción de la iteración**	**Orden de la HU a implementar**	**Duración total**[1]
1	Se obtiene la versión 0.1 del Detector de Rostros con la terminación de las HU asociadas.	IMG-11, IMG-12	5
2	Se obtiene la versión 0.1 del Identificador de Rostros con la terminación de las HU asociadas.	IMG-23, IMG-24, IMG-25, IMG-26	4

Continúa en la próxima página

[1] Estimado en semanas.

3	Se obtiene la versión 0.1 del Módulo de Reconocimiento de Rostros con la terminación de las HU asociadas.	IMG-27, IMG-28	2

Tabla 3.1 – Plan de *release*.

3.2. Diagrama de componentes

Los diagramas de componentes muestran las dependencias del compilador y del *runtime* entre los componentes del software; por ejemplo, los archivos del código fuente y las bibliotecas [24]. La Figura 3.2 se muestra una gran dependencia de los filtros con las bibliotecas OpenCV y mocic-common. Se observa además la necesidad de varios ficheros XML donde se almacenan los entrenamientos utilizados por los algoritmos tanto de detección como de identificación. La ejecución comienza en el componente start-stop-daemon, el cual inicia el ejecutable MOCIC-IMG, este primeramente carga la configuración de un fichero XML por medio de la biblioteca mocic-common, luego carga los filtros necesarios y finalmente ejecuta los filtros en el orden definido en el fichero de configuración. Cada filtro realiza su función y los ficheros de entrenamiento son creados o cargados teniendo en cuenta la tarea a realizar.

3.3. Diagrama de despliegue

Los diagramas de despliegue muestran a los nodos procesadores la distribución de los procesos y de los componentes [24]. En la Figura 3.1 se muestra el diagrama de despliegue del módulo de reconocimiento de rostros para MOCIC, el cual se ejecuta en un nodo independiente del módulo controlador, además se comunica con el depósito del cual toma las imágenes a procesar. La comunicación con el controlador se lleva cabo mediante el protocolo TCP/IP, que permite el intercambio de mensajes entre ambos módulos logrando así la integración con el resto del sistema. La comunicación con el depósito se realiza mediante el protocolo samba.

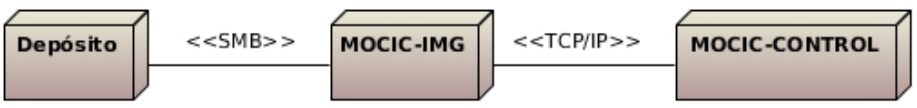


Figura 3.1 – Diagrama de despliegue.

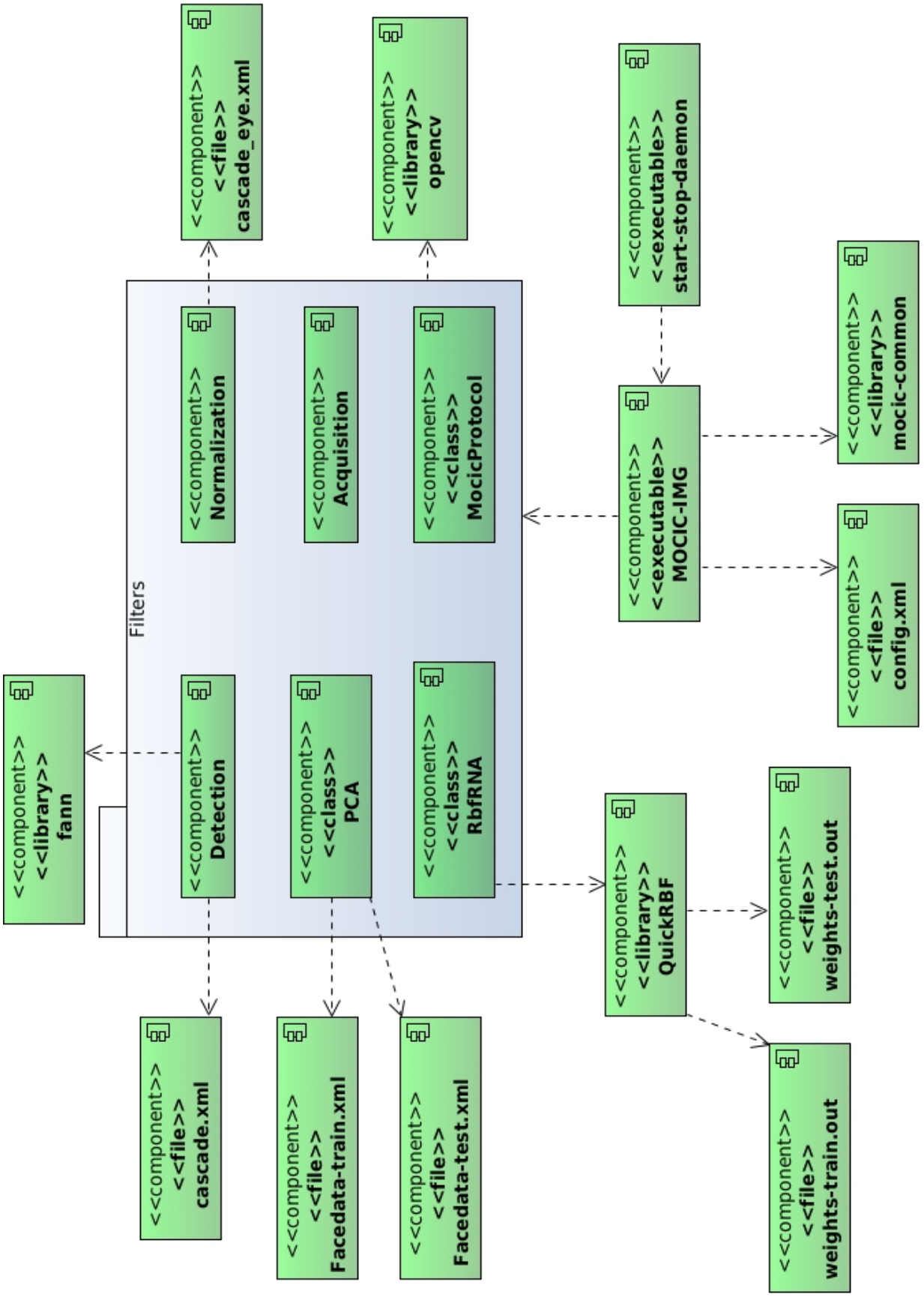


Figura 3.2 – Diagrama de componentes.

3.4. Diseño y ejecución de las Pruebas de Software

Con el fin de validar el correcto funcionamiento del módulo de reconocimiento de rostros para MOCIC se realizan pruebas de funcionalidad, efectividad e integración. A continuación se describe el proceso necesario para llevar a cabo cada una de ellas.

3.4.1. Pruebas de Funcionalidad

Las pruebas de funcionalidad se enfocan en las acciones por parte del usuario y las respuestas por parte del sistema, se llevan a cabo para comprobar los requerimientos y se considera que una funcionalidad tiene éxito cuando se comporta de la manera esperada por el cliente [32]. Para comprobar el cumplimiento de las funcionalidades del sistema se diseñaron casos de prueba para cada HU y a continuación se relacionan las más importantes. La Tabla 3.2 muestra el caso de prueba para la funcionalidad encargada de detectar rostros en una imagen, la cual consiste en cargar una imagen y localizar los rostros existentes por medio de un rectángulo que indica la posición de los mismos. En la Tabla 3.3 se describen las condiciones necesarias y los pasos a seguir para comprobar el correcto funcionamiento de la identificación de rostros, la cual consiste en devolver a quién pertenecen los rostros detectados.

Casos de Prueba de Funcionalidad	
Código Caso de Prueba: MOCIC- IMG-11 -1	**Nombre Historia de Usuario:** Detectar rostros en una imagen.
Nombre de la persona que realiza la prueba: Yoana Pita Lorenzo.	
Descripción de la Prueba: En una imagen dada se detectarán los rostros contenidos en ella, para buscar los Falsos Positivos y los Falsos Negativos.	
Condiciones de ejecución: Las imágenes a probar deben contener al menos un rostro y deben tener buenas condiciones de iluminación y una resolución de más de 100×100.	
Continúa en la próxima página	

Entrada/Pasos de ejecución: 1. Crear una estructura de carpeta (rostros a detectar, rostros detectados). 2. Modificar el fichero de configuración especificándole donde se encuentran las carpetas mencionadas y el fichero de entrenamiento.
Resultado Esperado: Se espera que devuelva un arreglo de rectángulos que señalan la localización y el tamaño de cada rostro con respecto a la imagen. Además se debe brindar la posibilidad de imprimir en consola alguna información de interés, como por ejemplo la cantidad de rostros detectados en la imagen.
Evaluación: Satisfactoria.

Tabla 3.2 – Caso de prueba de la funcionalidad: Detectar rostros en una imagen.

	Casos de Prueba de Funcionalidad
Código Caso de Prueba: MOCIC- IMG-23 -3	**Nombre Historia de Usuario:** Identificar rostros en una imagen.
Nombre de la persona que realiza la prueba: Yoana Pita Lorenzo.	
Descripción de la Prueba: Dado una imagen de un rostro se identificará a que persona pertenece.	
Condiciones de ejecución: La imagen que se utilice debe poseer buenas condiciones de iluminación y la resolución debe ser mayor de 100×100, para obtener resultados satisfactorios. Además para poder identificar una persona, esta debe estar en el entrenamiento.	
Entrada/Pasos de ejecución: 1. Especificar en el fichero de configuración el número de centros a utilizar. 2. Ejecutar el módulo, especificándole la dirección del fichero de configuración para el reconocimiento.	
Resultado Esperado: Devuelve el número de la persona a la cual pertenece el rostro que se desea reconocer y en caso de no reconocerlo devuelve -1.	
Evaluación: Satisfactoria.	

Tabla 3.3 – Caso de prueba de la funcionalidad: Identificar rostros en una imagen.

3.4.2. Pruebas de de Efectividad

En la literatura anglosajona consultada, las pruebas para determinar la efectividad de los algoritmos de clasificación automática se conocen como *performance testing* o pruebas de rendimiento, las cuales se han diseñado para probar el rendimiento en tiempo de ejecución de un software [32]. Dado que el módulo de reconocimiento de rostros para MOCIC utiliza técnicas de Inteligencia Artificial (IA), son necesarias las pruebas de efectividad para complementar las pruebas de funcionalidad del sistema. Por otra parte, algunos algoritmos requieren parámetros para su correcto funcionamiento; mediante estas pruebas se pueden establecer los parámetros que brindan mejores resultados. Con el fin de evaluar los resultados obtenidos, se definieron los criterios:

- Falso Positivo (FP): Ocurre cuando un elemento es enmarcado dentro de una clase conocida y este no pertenece a dicha clase.
- Falso Negativo (FN): Se evidencia cuando el sistema retorna que un elemento es desconocido y sin embargo pertenece una clase conocida.

Detector de rostros

Para las pruebas de efectividad de la detección de rostros se utilizó una muestra de 100 imágenes tomadas en un entorno no controlado, en las cuales varía la cantidad de rostros, las condiciones de iluminación, el fondo y el tamaño.

La efectividad es calculada mediante Fórmula 3.1, que se muestra a continuación:

$$PE = 100 - \frac{(FP + FN) * 100}{CR} \tag{3.1}$$

donde, CR es la cantidad de rostros a detectar y PE es el por ciento de efectividad de la detección.

La Tabla 3.4 muestra el resultado de la prueba realizada al detector de rostros, en la cual se obtuvo un 72.8 % de efectividad en la detección de rostros y está influenciado mayoritariamente por la ocurrencia de FP. La cantidad de FP se puede disminuir por medio de las características, relacionadas con los rostros, identificadas en la sección 2.1.1, lo cual significaría una mejora en la efectividad de la detección. Para comprobar lo propuesto en la sección 2.1.1, se realizaron pruebas al detector de rostros con la misma

colección de imágenes utilizada anteriormente. Primeramente, se probó descartando los posibles rostros por la cantidad de regiones de piel, luego por el *compactness* y finalmente combinando ambos métodos.

FP	FN	CR	PE
32	21	191	72.8%

Tabla 3.4 – Resultado con imágenes en un entorno no controlado.

Descartar por ambos métodos requiere establecer el rango de aceptación para que un rostro candidato no sea descartado. Se estableció, para cada caso, una aproximación inicial a dicho rango por medio una prueba con 200 rostros, a cada uno de los cuales se le extrajo la cantidad de regiones de piel y el *compactness* respectivamente.

Descartado por la cantidad de regiones de piel

Para establecer el rango inicial de la cantidad de regiones de piel se realizó la prueba y se obtuvo como resultado, que la mayor cantidad de rostros poseen entre 1 y 3 regiones de piel, como se observa en la Figura 3.3. No obstante, la prueba de efectividad de la detección arrojó como mejor resultado, 77.9% con el rango comprendido entre 1 y 6 regiones de piel, lo cual se muestra en la Tabla 3.5.

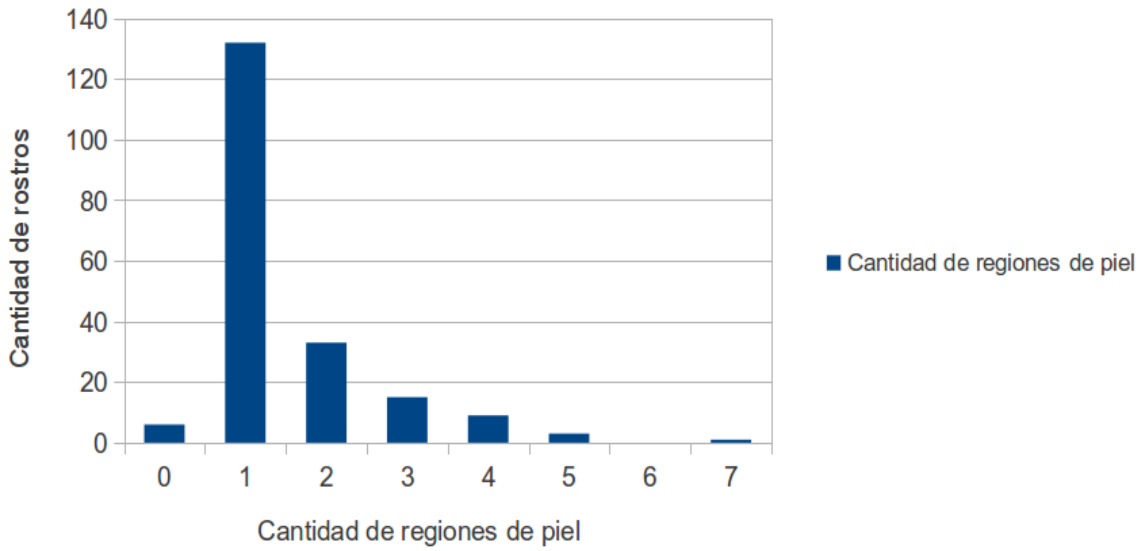


Figura 3.3 – Cantidad de regiones de piel.

FP	FN	CR	PE
21	22	195	77.9%

Tabla 3.5 – Mejores resultados obtenidos con la cantidad de regiones de piel.

Descartado por *compactness*

Para establecer el rango inicial de *compactness*, como se observa en la Figura 3.4, la mayor cantidad de rostros se encuentran dentro del rango [0.6 - 0.9].

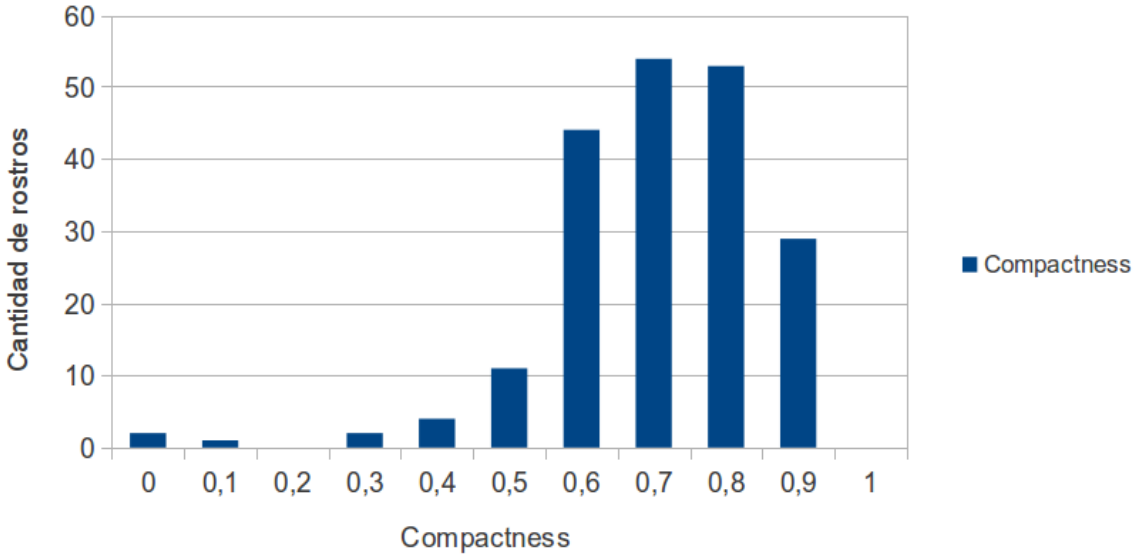


Figura 3.4 – Cantidad de rostros contenidos en cada rango del *compactness*.

A partir del rango de aceptación inicial se realizaron pruebas variando el mismo y los mejores resultados se obtuvieron en el rango de [0.3-0.9] con un 81% de efectividad, como se muestra en la Tabla 3.6.

FP	FN	CR	PE
10	27	195	81%

Tabla 3.6 – Mejores resultados obtenidos con el *compactness*.

Descartado por la cantidad de regiones de piel y por *compactness*

A partir de la realización de una prueba de efectividad que combina el descartado por la cantidad de regiones de piel y por *compactness*, se obtuvo como resultado una efectividad de 81.5%, ver Tabla 3.7.

FP	FN	CR	PE
11	25	195	81.5%

Tabla 3.7 – Descartado por *compactness* y por la cantidad de regiones de piel.

Discusión de los resultados de las pruebas realizadas al detector de rostros

Para un mejor entendimiento, se toma como convenio la notación: (1) para AdaBoost, (2) para el descartado por cantidad de regiones de piel y (3) para representar el descartado por *compactness*. Los resultados alcanzados en cada una de las pruebas se resumen en la Tabla 3.8.

Pruebas realizadas	FP	FN	CR	PE
1	32	21	195	72.8%
1-2	21	22	195	77.9%
1-3	10	27	195	81%
1-2-3	**11**	**25**	**195**	**81.5%**

Tabla 3.8 – Resultados de las pruebas realizadas al detector de rostros.

Luego de realizar las pruebas de efectividad al detector de rostros se concluye que: la combinación de ambos métodos es más efectiva que cada método por separado y se obtuvo un detector de rostros con una efectividad de 81.5%, la cual mejoró en un 8.7% con respecto a la efectividad inicial. Finalmente, el rango de aceptación del *compactness* que ofrece mejores resultados es de 0.3 a 0.9, y de la cantidad de regiones de piel es de 1 a 6.

Detector de ojos

Para las pruebas de efectividad de la detección de ojos se utilizó una muestra de 100 imágenes de rostros previamente detectados y extraídos de imágenes tomadas en un entorno no controlado, en las cuales varían las condiciones de iluminación, el fondo y el tamaño.

La efectividad es calculada mediante la fórmula 3.2 que se muestra a continuación:

$$PE = 100 - \frac{(FP + FN) * 100}{CO} \tag{3.2}$$

donde, CO es la cantidad de ojos a detectar y PE es el por ciento de efectividad de la detección de ojos.

El resultado de esta prueba se muestra en la Tabla 3.9 y está influenciado mayoritariamente por la ocurrencia de FP. La cantidad de FP, al igual que en la detección de rostros, pudiera disminuir por medio de 2 características relacionadas con el tamaño y posición de los ojos con respecto al rostro, las cuales se describen en la sección 2.1.2. Se realizaron pruebas al detector de ojos con la misma colección de imágenes utilizada anteriormente. Primeramente se probó descartando los posibles ojos por el tamaño, luego por la posición y por último se combinaron ambos métodos. El procedimiento llevado a cabo es similar al de la detección de rostros y se puede observar en el Anexo D.

FP	FN	CO	PE
34	16	200	75%

Tabla 3.9 – Resultado del detector de ojos.

Discusión de los resultados de las pruebas de efectividad al detector de ojos

La notación asumida para representar cada método es la siguiente: (1) para AdaBoost, (2) para el descartado por tamaño y (3) para el descartado por posición. Los resultados de las pruebas realizadas al detector de ojos se pueden observar en la Tabla 3.10.

Pruebas realizadas	FP	FN	CR	PE
1	34	16	200	75%
1-2	32	16	200	76%
1-3	21	16	200	81.5%
1-2-3	**20**	**16**	**200**	**82%**

Tabla 3.10 – Resultados de las pruebas de efectividad al detector de ojos.

Luego de realizar las pruebas de efectividad al detector de ojos se concluye que: la combinación de ambos métodos es más efectiva que cada método por separado. Se obtuvo un detector de ojos con una efectividad de 82% la cual mejoró en un 7% con respecto a la efectividad inicial de la detección de ojos. Por último, el rango de aceptación del tamaño relativo del ojo con respecto al rostro que ofrece mejores resultados es de 0.04 a 0.25, y para el caso de la posición del ojo con respecto al rostro es de 0.04 a 0.4.

Identificador de rostros

La efectividad del identificador de rostros ha sido evaluada con la base de datos de rostros de los Laboratorios AT&T de Cambridge[2], formalmente conocida como base de datos de rostros Olivetti Research Laboratories (ORL). Esta base de datos contiene 400 imágenes en escala de grises de 40 personas, a razón de 10 imágenes por persona. Las imágenes presentan pequeñas variaciones de iluminación, de orientación y de expresiones faciales[3].

La validación cruzada es un método estadístico para la evaluación de algoritmos de aprendizaje mediante la división de los datos en 2 partes, una para entrenar y la otra para validar el modelo [33]. La validación cruzada de k iteraciones[4] es la forma básica de la validación cruzada, en la cual los datos son divididos en k subconjuntos iguales, donde k-1 se utilizan para el entrenamiento y el subconjunto restante para la validación. Este proceso se realiza k iteraciones y en cada una cambian los subconjuntos de entrenamiento y de validación respectivamente. Un ejemplo de ello se observa en la Figura 3.5 donde k=3.

[2] http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

[3] Sonrisa, ojos abiertos o cerrados.

[4] Conocido como 10-fold cross validation en la bibliografía anglosajona.

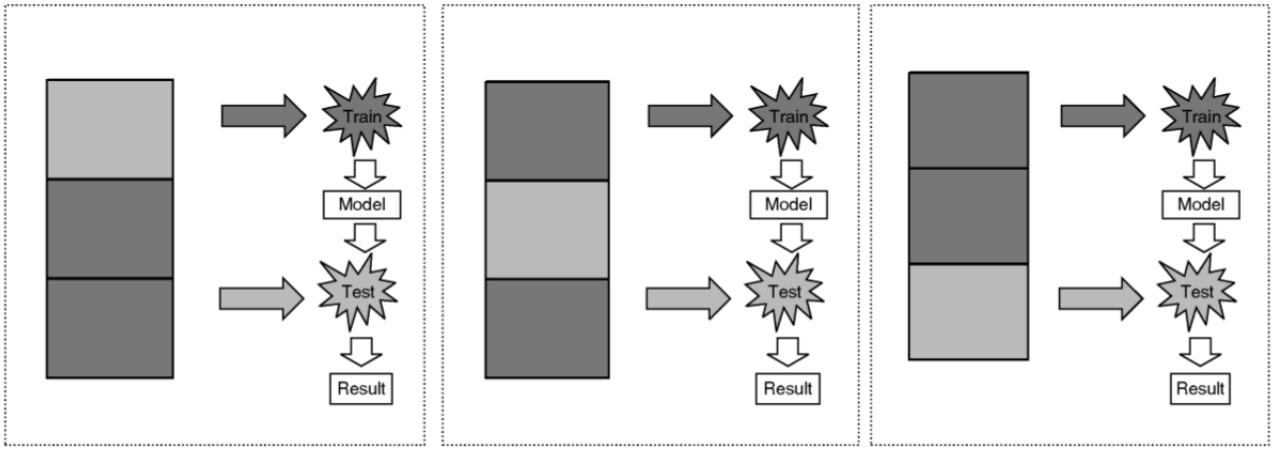


Figura 3.5 – Validación cruzada de k iteraciones, fuente: [33].

Para la realización de los experimentos se ha decidido utilizar el método de validación cruzada de 10 iteraciones[5], cuyo resultado final es la media de cada iteración y es válido señalar que el conjunto de prueba es equilibrado, dado que tiene la misma cantidad de personas conocidas que desconocidas.

La efectividad es calculada mediante la fórmula 3.3 que se muestra a continuación:

$$PE = \frac{(FP+FN)}{(FP+FN+TA)} * 100 \tag{3.3}$$

donde, TA es el total de aciertos y PE es el por ciento de efectividad de la identificación de rostros.

Las pruebas se realizaron con 20 neuronas en la capa oculta, una neurona para cada persona y con σ=3 para el entrenamiento y σ=2.25 para la prueba, dado que con esta topología de red se obtuvieron los mejores resultados.

Los resultados de la prueba realizada para determinar la cantidad de componentes principales a tener en cuenta en el algoritmo PCA se muestran a continuación en la Figura 3.6, y se pueden observar con más detalle en el Anexo E.

[5]Conocido como 10-fold cross validation en la bibliografía anglosajona.

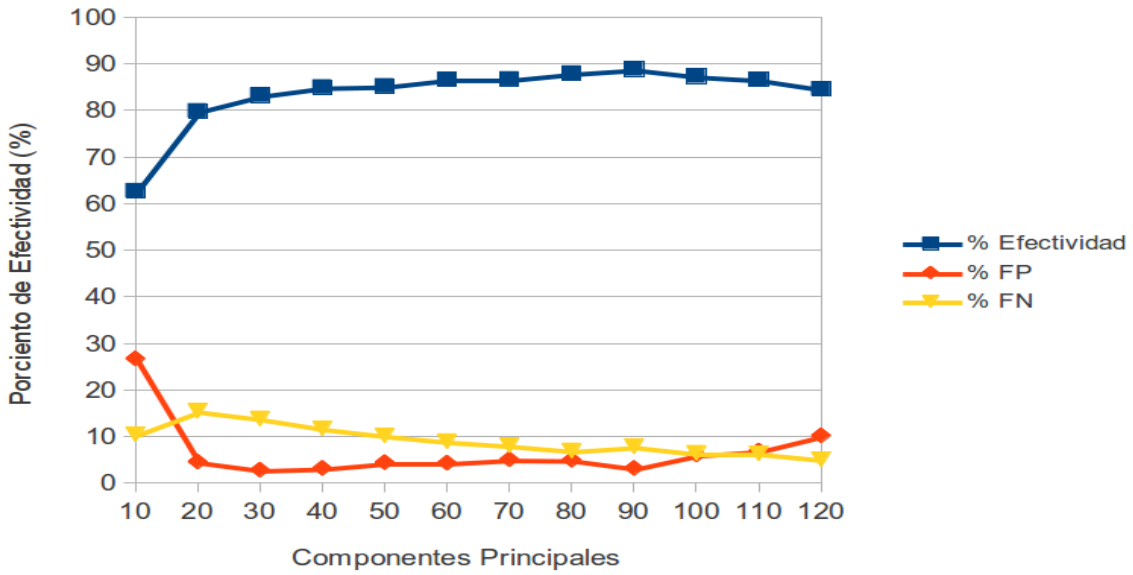


Figura 3.6 – Representación de los componentes principales y sus resultados.

Luego de realizar 10 iteraciones para cada uno de los valores de los componentes principales los mejores resultados se obtuvieron con 90, con una efectividad media de 88.9%. Además, se calculó el tiempo promedio en identificar una imagen en función de la cantidad de componentes principales, este experimento se realizó en una computadora Intel Dual Core a 3.20GHz, 2Gb DDR3 con el sistema operativo Ubuntu 12.10. Los resultados se muestran en la Figura 3.7, de la cual se concluye que el tiempo de identificación aumenta linealmente con el número de componentes principales; y para 90 *eigenfaces* el módulo tarda aproximadamente un tercio de segundo en identificar una persona.

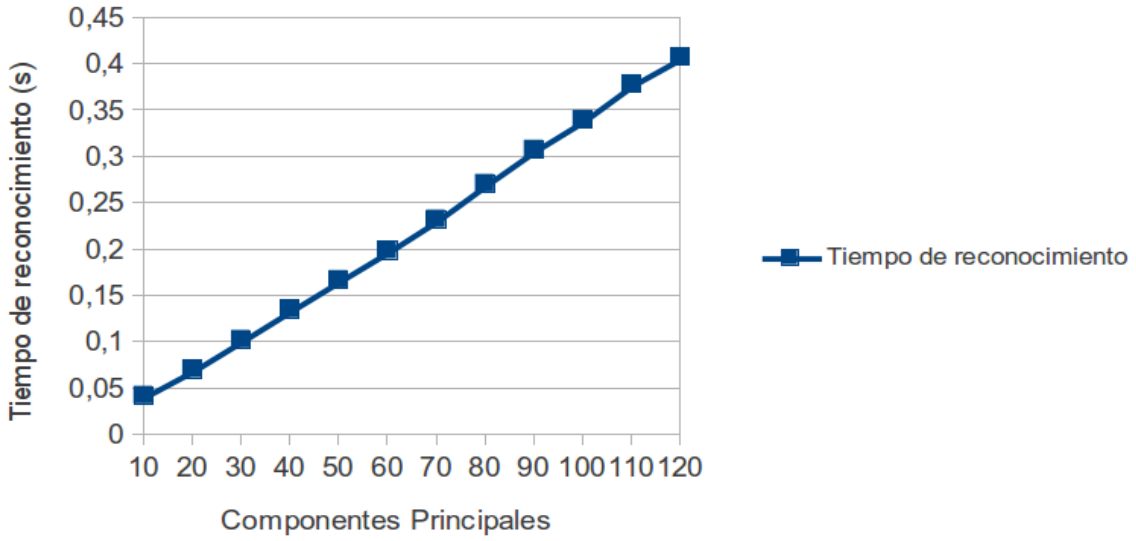


Figura 3.7 – Tiempo de reconocimiento.

Seguidamente, en la Figura 3.8 se muestran los resultados de la prueba para determinar el umbral de aceptación que maximice la efectividad del identificador de rostros. En el Anexo E se puede encontrar más información.

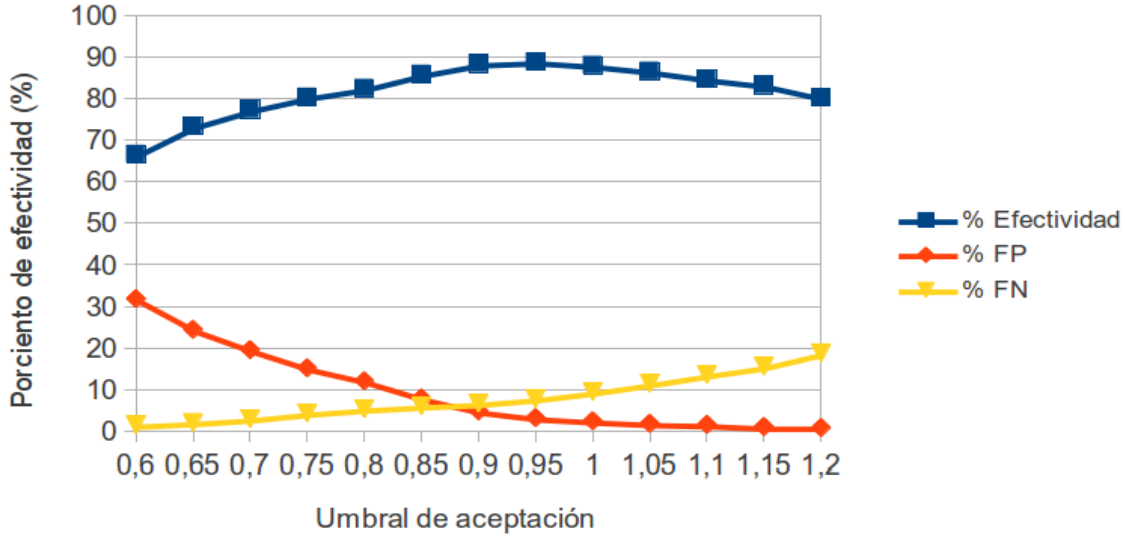


Figura 3.8 – Representación de cada umbral de aceptación y sus resultados.

Luego de analizar los resultados del experimento se concluye que para un umbral de 0.95 la efectividad

es máxima con una media de 88.9%, aunque se obtuvieron buenos resultados en el rango comprendido entre 0.9 y 1.

Finalmente, se concluye que se obtuvo un identificador de rostros con una efectividad de 88.9% con los siguientes parámetros: 90 componentes principales para el algoritmo PCA; 40 neuronas en la capa oculta de la Red Neuronal de Función de Base Radial; σ=3 para el entrenamiento, σ =2.25 para el reconocimiento y 0.95 como umbral de aceptación.

3.4.3. Pruebas de Integración

Son una técnica de realización de pruebas para descubrir los errores asociados con la interconexión. El objetivo es tomar los componentes y crear un programa para detectar posibles pérdidas de datos entre ellos [32].

La integración se lleva a cabo a través del módulo controlador, que envía un mensaje a los restantes módulos, los cuales le responden con los parámetros correspondientes. Para realizar la prueba de integración, primeramente hay que establecer la conexión, lo cual requiere conocer la dirección IP de la computadora donde se encuentra el módulo controlador y el puerto de comunicación. Una vez conectado, el módulo controlador envía un mensaje con la dirección de la carpeta que contiene las imágenes a procesar. Luego de reconocer los rostros, el sistema envía una respuesta al controlador con la categoría de cada imagen. El sistema categoriza una imagen teniendo en cuenta las categorías de los rostros que posee, donde la que predomina se toma como categoría de la imagen.

A continuación se detalla la estructura de la respuesta que se envía al módulo controlador:

-n IMG -r X : X : X : ... : X

Descrita por los parámetros:

- *-n*: Nombre del módulo que envía el mensaje.
- *-r*: Respuesta del módulo.
- X: Cada X representa una imagen y el valor de la X es el identificador de su categoría. El rango admitido por X es un número entero de [0-8]. Cuando $X = 0$, significa que no se pudo enmarcar la

imagen en ninguna de las categorías.

Conclusiones

Una vez implementado el sistema y validado su funcionamiento, se concluye que:

- La realización de los diagramas de componentes y despliegue posibilitaron un mejor entendimiento de la interacción de los diferentes componentes del software, ayudando así a su implementación.
- El lenguaje C++ constituyó una buena elección para la implementación del sistema.
- Un correcto diseño de las pruebas contribuyó a establecer los parámetros para maximizar la efectividad de los algoritmos, así como posibilitó la realización a tiempo de acciones correctivas para garantizar la calidad del software.

Conclusiones

Con la culminación del presente trabajo de diploma se concluye que:

- Las aplicaciones y herramientas que incluyen el reconocimiento de rostros se insertan cada vez más en la vida diaria, con marcado predominio de software privativo, lo cual dificulta su utilización en nuestro país.
- Con la implementación del módulo de reconocimiento de rostros, MOCIC puede detectar e identificar el rostro de una persona en una imagen digital.
- Luego de implementar el módulo de reconocimiento de rostros se validó su funcionamiento, obteniendo un detector de rostros con una efectividad de 81.5% y un identificador con una efectividad de 88.8%, además se logró la integración con MOCIC de manera satisfactoria.

Por lo cual, los resultados obtenidos, dan cumplimiento a cada uno de los objetivos propuestos.

Recomendaciones

Para mejorar el sistema, el despliegue del mismo y seguir profundizando en el campo del reconocimiento facial, se recomienda:

- Añadir el algoritmo LDA en la etapa de Representación.
- Probar el módulo con otras colecciones de entrenamiento para comprobar su efectividad.
- Utilizar la técnica *Elastic Bunch Graph Matching* para la identificación de rostro.
- Implementar una interfaz gráfica para el uso de la herramienta por usuarios no avanzados.
- Aplicar el módulo implementado para adicionar un descriptor al módulo de detección de imágenes desnudas.

Referencias bibliográficas

[1] Kiuver Kaddiel Ibañez Castro. *Diseño e implementación de un categorizador automático de imágenes pornográficas.* PhD thesis, Universidad de las Ciencias Informáticas, 2008.

[2] Avalon Biometrics SL. Avalon biometrics [en línea]. 2007. Disponible en: http://www.avalonbiometrics.com/who_are_we.html [Accedida 18 de noviembre de 2011].

[3] TAB Systems. Smarti [en línea]. 2012. Disponible en: http://www.tab-systems.com/ [Accedida 18 de noviembre de 2011].

[4] Pittsburgh Pattern Recognition, Inc. Pittpatt [en línea]. 2012. Disponible en: http://www.pittpatt.com/ [Accedida 18 de noviembre de 2011].

[5] Airborne Biometrics Group, Inc. facefirst [en línea]. 2011. Disponible en: http://www.facefirst.com/abgOverview.html [Accedida 18 de noviembre de 2011].

[6] face.com. face.com [en línea]. 2012. Disponible en: http://developers.face.com/docs/ [Accedida 18 de noviembre de 2011].

[7] PAM Face Developers. Pluggable authentication module for face authentication. [en línea]. 2012. Disponible en: http://pam-face-authentication.org/ [Accedida 18 de noviembre de 2011].

[8] Yoel Hernández Mendoza Yadier Díaz Cárdenas. *Incorporación de Identificación Biométrica al Sistema CERES.* PhD thesis, Universidad de las Ciencias Informáticas, 2009.

[9] E. Cabello Pardos. *Técnicas de reconocimiento facial mediante redes neuronales.* PhD thesis, Facultad de Informática (UPM), 2004. pp. 18-20.

[10] Maria Victoria Fornaguera Rodríguez. *Propuesta de un sistema de reconocimiento de rostros para MOCIC.* PhD thesis, Universidad de las Ciencias Informáticas, 2009.

[11] Gimeno Hernández. Estudio de técnicas de reconocimiento facial. 2011.

[12] P. Viola and M.J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[13] N. Degtyarev and O. Seredin. Comparative testing of face detection algorithms. *Image and Signal Processing*, pages 200–209, 2010.

[14] R.C. Gonzalez and R.E. Woods. Digital image processing second edition. *Beijing: Publishing House of Electronics Industry*, 2002.

[15] K. Delac and M. Grgic. Face recognition, i-tech education and publishing. *Vienna, Austria*, 2007.

[16] Universidad de Los Andes Mérida Venezuela. Pagina de gerardo colmenares [en línea]. 2001. Disponible en: http://webdelprofesor.ula.ve/economia/gcolmen/postgrado1.html [Accedida 29 de febrero de 2012].

[17] S. Thakur, JK Sing, DK Basu, M. Nasipuri, and M. Kundu. Face recognition using principal component analysis and rbf neural networks. In *Emerging Trends in Engineering and Technology, 2008. ICETET'08. First International Conference on.*, pages 695–700. IEEE, 2008.

[18] N.M. Josuttis. *The C++ standard library: a tutorial and reference.* Addisonb Wesley Longman, Inc, 1999.

[19] Yu-Yen Ou. QuickRBF: an efficient RBFN package. [en línea]. 2012. Disponible en: http://csie.org/~yien/quickrbf/index.php [Accedida 15 de febrero de 2012].

[20] G. Bradski and A. Kaehler. Learning opencv: Computer vision with the opencv library. *Learning*, 2008.

[21] Jeffrey D. Ullman. Alfred V. Aho, Ravi Sethi. *Compilers, principles, techniques, and tools.* Greg Tobin, 1986.

[22] Oracle Corporation and/or its affiliates. Getting started with an integrated development environment [en línea]. 2010. Disponible en: http://java.sun.com/developer/technicalArticles/tools/intro.html [Accedida 6 de junio 2012].

[23] Gladys Marsi Peñalver Romero. *MA-GMPR-UR2 Metodología ágil para proyectos de software libre.* PhD thesis, Universidad de las Ciencias Informáticas, 2008.

[24] C. Larman. Uml y patrones. introducción al análisis y diseño orientado a objetos. *Prentice-Hall*, 2002.

[25] I. Sommerville. *Software Engineering 8th Revised edition.* Addison-Wesley Educational Publishers Inc, United Kingdom., 2006.

[26] Visual Paradigm Company. Visual paradigm for uml 8.3 model-code-deploy platform [en línea]. 2012. Disponible en: http://www.visual-paradigm.com/product/vpuml/provides/umlmodeling.jsp [Accedida 10 de febrero de 2012].

[27] Corporación Instituto para La Educación Pastorial. PNG Imagen [en línea]. 2012. Disponible en: http://www.cipep.com/imagenes/3people.png [Accedida 20 de febrero de 2012].

[28] T. Acharya and A.K. Ray. *Image processing: principles and applications.* Wiley-Interscience, 2005.

[29] K. Beck and M. Fowler. *Planning extreme programming.* Addison-Wesley Professional, 2001.

[30] F. Buschmann, K. Henney, and D.C. Schmidt. *Pattern-oriented software architecture: On patterns and pattern languages.*, volume 5. John Wiley & Sons Inc, 2007.

[31] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Elements of reusable object-oriented languages and systems., 1994.

[32] R.S. Pressman. *Software Engineering A Practitioner's Approach.* McGraw Hill Publication, 2010.

[33] Huan Lui Payam Refaeilzadeh, Lei Tang. K-fold cross-validation. *Arizona State University*, 2008.

Bibliografía

- Mislav Grgic Kresimir Delac and Marian Stewart Bartlett. Recent adavantages in face recognition, i-tech. *Vienna, Austria*, 2008.
- I. Sommerville. *Software Engineering 8th Revised edition*. Addison-Wesley Educational Publishers Inc, United Kingdom, 2006.
- R.S. Pressman. *Software Engineering A Practitioner's Approach*. McGraw Hill Publication, 2010.
- J. Schmuller. *Aprendiendo UML en 24 horas*. Pearson educación, 2000.
- R. Laganière. *OpenCV 2 computer vision application programming cookbook*. Packt Publ. Limited, 2011.
- S. Kottwitz. *LaTeX Beginner's Guide*. Packt Publishing, 2011.
- D. Jyoti, A. Chadha, P. Vaidya, and M.M. Roja. A robust, low-cost approach to face detection and face recognition. *Arxiv preprint arXiv:1111.1090*, 2011.
- C. Zhang and Z. Zhang. A survey of recent advances in face detection. *Microsoft Research, June*, 2010.
- Hyun-Cheol Choi Unsang Park, Brendan Klare. Automatic face recognition state of the art aniljain.
- V.B. Rao and H. Rao. *C++, neural networks and fuzzy logic*. Mis: Press, 1995.
- N. Degtyarev and O. Seredin. Comparative testing of face detection algorithms. *Image and Signal Processing*, pages 200–209, 2010.
- C.E. Thomaz, R.Q. Feitosa, and A. Veiga. Design of radial basis function network as classifier in face recognition using eigenfaces. In *Neural Networks, 1998. Proceedings. Vth Brazilian Symposium on*, pages 118–123. IEEE, 1998.
- Edgar Acuna. Métodos para estimar el error de predicción en regresión: Validación cruzada y boostrapping.
- R. Gimeno Hernández. Estudio de técnicas de reconocimiento facial. 2011.

- C. Elkan. Evaluating classifiers. 2007.
- Kimberly Weissman. Face recognition.
- S.R. Baragada, S. Ramakrishna, MS Rao, and S. Purushothaman. Implementation of radial basis function neural network for image steganalysis. *International Journal of Computer Science and Security (IJCSS)*, 2(1):12, 2008.
- A.P. Kumar, TN Rickesh, R.V. Babu, and R. Hariharan. Object tracking using radial basis function networks.
- Javier Molina Cantero. Prácticas de laboratorio. Práctica 3: Realce de imágenes.
- P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.
- P. Courrieu. Straight monotonic embedding of data sets in euclidean spaces. *Neural Networks*, 15(10):1185–1196, 2002.
- Radial basis function neural network tutorial.
- M. Delbracio and M. Mateu. Trabajo final de reconocimiento de patrones: Identificación utilizando PCA, ICA y LDA.
- Poggio y Girosi. Redes neuronales con base radial, 1990.
- R. Collobert, S. Bengio, and J. Mariéthoz. Torch: a modular machine learning software library. Technical report, Technical Report IDIAP-RR 02-46, IDIAP, 2002.
- J. Yang, D. Zhang, A.F. Frangi, and J. Yang. Two-dimensional pca: a new approach to appearance-based face representation and recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(1):131–137, 2004.
- J. Park and I.W. Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.
- P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2001.

- M. Agarwal, H. Agrawal, N. Jain, and M. Kumar. Face recognition using principle component analysis, eigenface and neural network. In *Signal Acquisition and Processing, 2010. ICSAP'10. International Conference on*, pages 310–314. IEEE, 2010.
- K.C. Jondhale and LM Waghmare. Improvement in pca performance using fld and rbf neural networks for face recognition. In *Emerging Trends in Engineering and Technology (ICETET), 2010 3rd International Conference on*, pages 500–505. IEEE, 2010.
- Lihi Zelnik-Manor Tal Hassner. Principal component analysis, eigenfaces and fisher discriminant.
- Filomen Incahuanaco Quispe Carlos Herrera Muñoz, Alfonso Phocco Diaz. Propuesta de implementación de un sistema de reconocimiento de expresiones faciales basado en flujo óptico.
- Ignacio Ramirez. Herramientas de autodocumentación y doxygen, octubre 2002.
- Roberto Albornoz. Manual de doxygen, 2003.
- Documentación automática con doxygen, abril 2008.

Anexo A

Tabla de la encuesta

Estadísticas sobre cantidad de documentos y su espacio ocupado en Gigabytes (Gbs).

Laboratorio	Cantidad de personas entrevistadas	Tamaño (Gb)
1	14	95
2	2	3
3	12	76
4	7	66
7	6	30
8	15	151
9	11	71
15	12	17
17	9	42
18	13	136
Total	**101**	**687**

Leyenda:

Esta encuesta fue realizada en los laboratorios de producción del Docente de Producción en la UCI.

Para el cálculo del tiempo aproximado que le toma a una persona categorizar un documento se tomó el tiempo que demora:

- Buscar el documento.
- Abrir el documento.
- Leer las primeras páginas hasta encontrar su categoría.
- Cerrar el documento.
- Guardar el documento en la categoría correspondiente.

Este proceso se realizó de forma ininterrumpida para una cantidad de 10 documentos seleccionados aleatoriamente, obteniéndose como tiempo total 6 minutos 32 segundos, dividiendo el tiempo total entre

la cantidad de documentos categorizados se obtiene el tiempo promedio que demoró la categorización de un documento. Los cálculos quedaron de la siguiente forma:

$$6min36s = 396s$$

$$396s/10 = 39{,}6s$$

Como el tiempo promedio en categorizar un documento que se obtuvo como resultado del experimento fue 39.6 segundos se tomó, para facilitar los cálculos posteriores, el valor aproximado de 40 segundos.

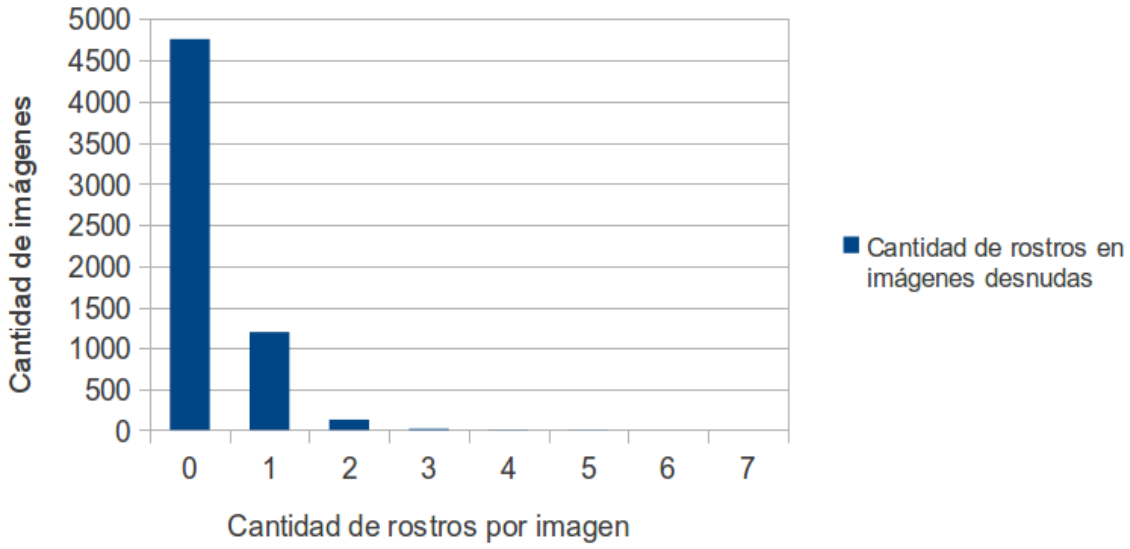
Anexo B

Estudio acerca de la cantidad de rostros en imágenes desnudas.

Para llevar a cabo el estudio del comportamiento de la cantidad de rostros en imágenes desnudas se utilizó una muestra de las imágenes de 100 sitios pornográficos de una colección de entrenamiento y el total de imágenes analizadas fue de 6079. A continuación se muestra la distribución de la cantidad de rostros por cada imagen:

Cantidad de rostros por imagen.

Cantidad de rostros	Cantidad de imágenes
0	4753
1	1184
2	125
3	13
4	3
5	1
6	0
7	0
Total	**6079**



Como se puede apreciar, en la Figura anterior, la mayoría de las imágenes contienen entre 0 y 1 rostro, aunque se detectaron hasta 5; para una media de 0.17 rostros en cada imagen, lo que denota la baja probabilidad de las imágenes desnudas de contener muchos rostros.

Anexo C

Descripción de las clases.

Descripción de la clase Filter.

Nombre de la Clase: Filter	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`img`	`object`
`ros`	`object`
`save`	`bool`
`print`	`bool`
`path_save`	`string`
`image_name`	`string`
`person_name`	`string`
`person_number`	`int`
`line`	`string`
`_call`	`ListString*`
`_next`	`ListString*`
Para cada responsabilidad	
Nombre:	`bool convertStringToBool(string _bool, string path)`
Descripción:	Convierte la cadena pasada por parámetro a el tipo de dato `bool`, solamente admite las cadenas true o false.
Nombre:	string get_image_name(`string path`)
Descripción:	Devuelve el nombre de la imagen que se encuentra en la dirección que se encuentra en `path`.
Nombre:	`string reverse_string(string str)`
Descripción:	Invierte el orden de la cadena pasada por parámetro.
Nombre:	`Filter()`
Descripción:	Constructor de la clase, devuelve un objeto Filter.

Continúa en la próxima página

Nombre:	`~Filter()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`void load()`
Descripción:	Carga y convierte las etiquetas comunes para todos los filtros, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object next(Object argument)`
Descripción:	Hace una llamada al filtro que le sigue en el fichero de configuración.

Descripción de la clase MocicProtocol.

Nombre de la Clase: MocicProtocol	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`data`	`Data`
`name`	`string`
`ip`	`string`
`port`	`string`
`protocol`	`Protocol`
Para cada responsabilidad	
Nombre:	`MocicProtocol()`
Descripción:	Constructor de la clase, devuelve un objeto MocicProtocol.
Nombre:	`~MocicProtocol()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`

Continúa en la próxima página

Descripción:	Establece una conexión con el módulo controlador, del cual recibe la dirección donde se encuentran las imágenes a procesar, envía la dirección al filtro Acquisition para procesar la imágenes y recibe de este, la categoría de las misma, luego envía la respuesta con la categoría de las imágenes al controlador.

Descripción de la clase PreProcess.

Nombre de la Clase: PreProcess	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`equalize`	`bool`
`resize`	`bool`
`size`	`int`
Para cada responsabilidad	
Nombre:	`PreProcess()`
Descripción:	Constructor de la clase, devuelve un objeto PreProcess().
Nombre:	`~PreProcess()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Reduce una imagen manteniendo sus proporciones de ancho por alto y ecualiza el histograma de la imagen. Recibe una imagen y devuelve una imagen preprocesada.
Nombre:	`histeq(IplImage* src, IplImage* dst)`

Continúa en la próxima página

Descripción:	Convierte el modelo de color de una imagen RGB a HSV y luego ecualiza el histograma de la imagen para lograr una iluminación uniforme de la misma.

Descripción de la clase ExtracImage.

Nombre de la Clase: ExtracImage	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`gray`	`bool`
`size`	`int`
`normalize`	`bool`
Para cada responsabilidad	
Nombre:	`ExtracImage()`
Descripción:	Constructor de la clase, devuelve un objeto ExtracImage().
Nombre:	`~ExtracImage()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Extrae cada rostro detectado y lo convierte en una imagen independiente de la original. Resolviendo el problema que surge cuando en una imagen son detectados más de un rostro.

Descripción de la clase segmentRNA.

Nombre de la Clase: segmentRNA
Tipo de Clase: Controladora
Continúa en la próxima página

Atributo	**Tipo**
`treshold`	`float`
`trainin`	`string`
`segment`	`string`
Para cada responsabilidad	
Nombre:	`RBFRNA()`
Descripción:	Constructor de la clase, devuelve un objeto RBFRNA.
Nombre:	`~RBFRNA()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Segmenta una imagen por el color de la piel y devuelve una imagen binaria donde los píxeles de valor 1 corresponden a piel y los de valor 0 no corresponden a regiones de piel.

Descripción de la clase Compactness.

Nombre de la Clase: Compactness	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`tresh`	`int`
`grather_than`	`int`
Para cada responsabilidad	
Nombre:	`Compactness()`
Descripción:	Constructor de la clase, devuelve un objeto *Compactness*.
Nombre:	`~Compactness()`

Continúa en la próxima página

Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Calcula el *compactness* de una imagen previamente segmentada y devuelve el valor calculado, el cual es un número real entre 0 y 1.

Descripción de la clase CountSkinRegions.

Nombre de la Clase: CountSkinRegions	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`tresh`	`int`
Para cada responsabilidad	
Nombre:	`CountSkinRegions()`
Descripción:	Constructor de la clase, devuelve un objeto CountSkinRegions.
Nombre:	`~CountSkinRegions()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Calcula la cantidad de regiones con piel de una imagen previamente segmentada y devuelve esa cantidad.

Descripción de la clase DiscardbyCompactenss.

Nombre de la Clase: DiscardbyCompactenss	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`treshmin`	`double`
`treshmax`	`double`
`draw_rectangle`	`bool`
Para cada responsabilidad	
Nombre:	`DiscardbyCompactenss()`
Descripción:	Constructor de la clase, devuelve un objeto DiscardbyCompactenss.
Nombre:	`~DiscardbyCompactenss()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(bject argument)`
Descripción:	Descarta los rostros detectados si el valor del *compactness* está fuera de un rango especificado.

Descripción de la clase DiscardbySize.

Nombre de la Clase: DiscardbySize	
Tipo de Clase: Controladora	
Atributo	**Tipo**
tresh	`double`
draw_rectangle	`bool`
Para cada responsabilidad	
Nombre:	`DiscardbySize()`

Continúa en la próxima página

Descripción:	Constructor de la clase, devuelve un objeto DiscardbySize.
Nombre:	`~DiscardbySize()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas.Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Descarta los rostros detectados si su tamaño con respecto al total de la imagen es menor que un umbral especificado.

Descripción de la clase DiscardbyRegions.

Nombre de la Clase: DiscardbyRegions	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`draw_rectangle`	`bool`
Para cada responsabilidad	
Nombre:	`DiscardbyRegions()`
Descripción:	Constructor de la clase, devuelve un objeto DiscardbyRegions.
Nombre:	`~DiscardbyRegions()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`

Continúa en la próxima página

Descripción:	Descarta los rostros detectados si la cantidad de regiones continuas con piel en ellos está fuera del rango especificado.

Descripción de la clase DiscardbySizeEye.

Nombre de la Clase: DiscardbySizeEye	
Tipo de Clase: Controladora	
Atributo	**Tipo**
`treshmin`	`double`
`treshmax`	`double`
`draw_rectangle`	`bool`
Para cada responsabilidad	
Nombre:	`DiscardbySizeEye()`
Descripción:	Constructor de la clase, devuelve un objeto DiscardbySizeEye
Nombre:	`~DiscardbySizeEye()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Descarta los ojos detectados si su tamaño con respecto al tamaño del rostro está fuera del rango especificado.

Descripción de la clase DiscardbyPositionEye.

Nombre de la Clase: DiscardbyPositionEye
Tipo de Clase: Controladora
Continúa en la próxima página

Atributo	**Tipo**
`treshmin`	`double`
`treshmax`	`double`
`draw_rectangle`	`bool`
Para cada responsabilidad	
Nombre:	`DiscardbyPositionEye()`
Descripción:	Constructor de la clase, devuelve un objeto DiscardbyPositionEye.
Nombre:	`~DiscardbyPositionEye()`
Descripción:	Destructor de la clase, borra las variables dinámicas.
Nombre:	`std::string objectId()`
Descripción:	Devuelve su nombre para construir una instancia de sí misma.
Nombre:	`void load()`
Descripción:	Carga y convierte sus etiquetas específicas, de cadenas de caracteres a los tipos de datos requeridos por cada una de ellas. Estas etiquetas se encuentran en el fichero de configuración.
Nombre:	`Object getOutput(Object argument)`
Descripción:	Descarta los ojos detectados si su localización con respecto al rostro está fuera de un rango especificado.

Anexo D

Pruebas para descartar los ojos candidatos por el tamaño y la posición con respecto al rostro.

Para descartar por ambos métodos es necesario establecer el rango de aceptación para que un ojo candidato no sea descartado. Se estableció, para cada caso, una aproximación inicial a dicho rango por medio del análisis de 300 ojos. Para cada ojo se calculó su tamaño y su posición.

Descartado por tamaño

El rango inicial de tamaño con respecto al rostro dentro del que se encuentran los ojos es de 0.04-0.1, ver Figura D.1.

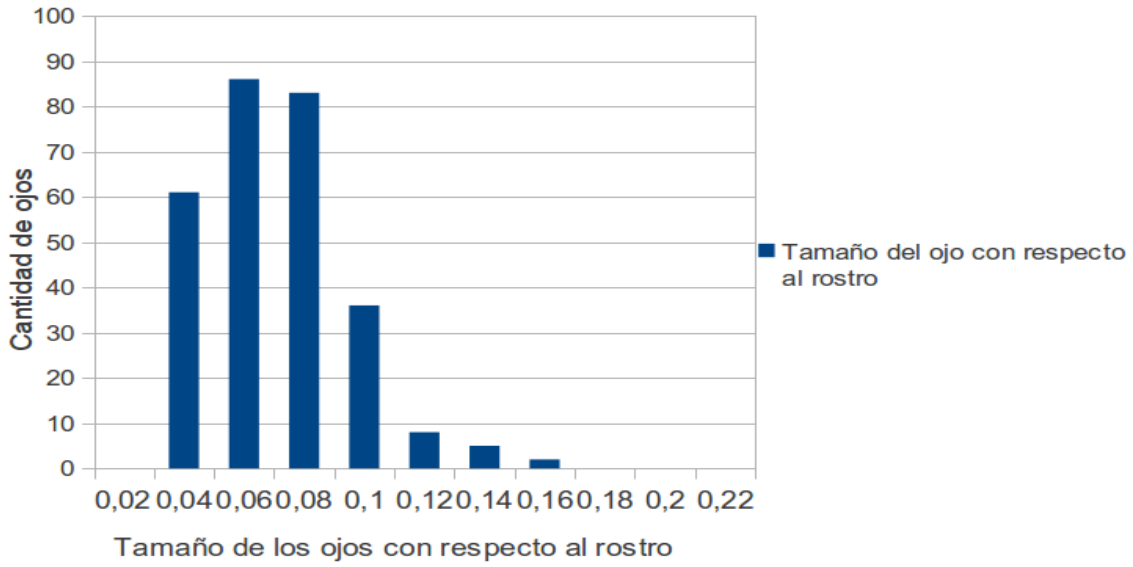


Figura D.1 – Tamaño de los ojos con respecto al rostro.

Las pruebas realizadas arrojaron los mejores resultados con el rango de 0.04 a 0.25, con un 76% de efectividad, como se muestra en la Tabla D.1.

FP	FN	CO	PE
32	16	195	76%

Tabla D.1 – Resultado de descartar por tamaño del ojo.

Descartado por posición

Luego de realizar la prueba se obtiene como resultado que la mayoría de los ojos tienen una posición relativa al rostro entre 0.2 y 0.35 con respecto al borde superior, como se observa en la Figura D.2.

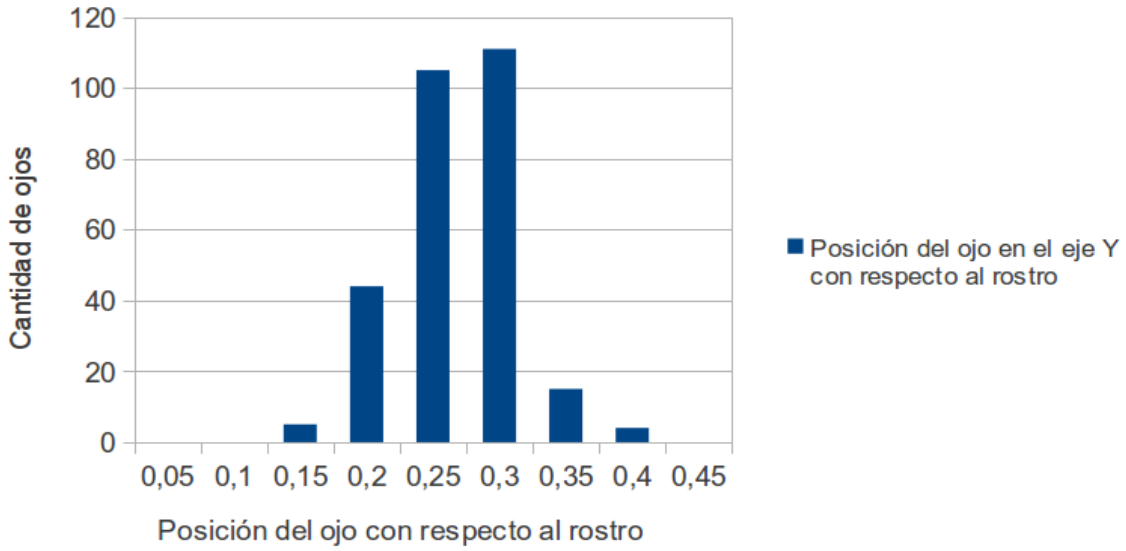


Figura D.2 – Posición del ojo con respecto al rostro.

A pesar del rango inicial seleccionado, la prueba de efectividad de la detección arrojó como mejor resultado 81.5% con el rango comprendido entre 0.04 y 0.4, como se muestra a continuación en la Tabla D.2.

FP	FN	CO	PE
21	16	200	81.5%

Tabla D.2 – Resultado de efectividad de la detección.

Descartado por tamaño y posición con respecto al rostro

Para finalizar se llevó a cabo una prueba que combina el descartado por tamaño y por posición del ojo con respecto al rostro, se obtuvo como resultado una efectividad de 82%, ver la Tabla D.3.

FP	FN	CO	PE
20	16	200	82%

Tabla D.3 – Resultado de descartar por tamaño y posición del ojo.

Anexo E

Pruebas del identificador de rostros.

A continuación los resultados de la prueba realizada para determinar la cantidad de componentes principales a tener en cuenta en el algoritmo PCA.

Componentes principales	% Efectividad	% FP	% FN
10	62.75	26.75	10.5
20	79.875	4.625	15.5
30	83.25	2.875	13.875
40	85	3.25	11.75
50	85.25	4.5	10.25
60	86.625	4.375	9
70	86.75	5.125	8.125
80	88	5	7
90	88.875	3.25	7.875
100	87.375	6.125	6.5
110	86.625	7	6.375
120	84.625	10.25	5.125

Tabla E.1 – Componentes principles y sus resultados.

Umbral de aceptación	% Efectividad	% FP	% FN
0.6	66.625	31.875	1.5
0.65	73.375	24.5	2.125
0.7	77.375	1.625	3
0.75	80.375	15.25	4.375
0.8	82.5	12.125	5.375
0.85	85.875	8	6.125
0.9	88.375	4.875	6.75
Continúa en la próxima página			

0.95	88.875	3.25	7.875
0.1	88	2.5	9.5
0.05	86.625	1.875	11.5
1.1	84.75	1.625	13.625
1.15	83.25	1.125	15.625
1.2	80.25	0.875	18.875

Tabla E.2 – Umbral de aceptación y sus resultados.

Acrónimos

MOCIC Motor de Categorización Inteligente de Contenido:
Es un proyecto de I+D cuyo objetivo principal es categorizar, de forma automática, documentos de diversos formatos.

Web Red Informática:
La traducción literal de esta palabra en inglés es telaraña, pero en términos informáticos se refiere al documento situado en una red informática, al que se accede mediante enlaces de hipertexto.

URL Uniform Resource Locator:
Localizador de Recursos Uniforme. Es una secuencia de caracteres, de acuerdo con un formato modélico y estándar, que se utiliza para nombrar recursos en Internet para su localización o identificación, como por ejemplo documentos textuales, imágenes, vídeos, en una página Web.

UCI Universidad de las Ciencias Informáticas:
La Universidad de las Ciencias Informáticas es una universidad creada al calor de la batalla de ideas con el propósito de forjar a las nuevas generaciones como Ingenieros en Ciencias Informáticas.

CAT Categorizador Automático de Texto.

CAI Categorizador Automático de Imágenes.

MD Módulo Decisor.

MRO Módulo de Reconocimiento de Objetos.

MDID Módulo de Detección de Imágenes Desnudas.

CCTV Circuito Cerrado de Televisión:
Es una tecnología de videovigilancia, diseñada para supervisar ambientes y actividades.

IP Internet Protocol:
Protocolo de Internet, es un protocolo no orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red.

API Application Programming Interface:
Interfaz de programación de aplicaciones, es el conjunto de funciones y procedimientos o métodos, en la programación orientada a objetos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

PIN Personal Identification Number:
Número de Identificación Personal, es una contraseña o clave numérica que se utiliza para acceder a cajeros automáticos o servicios de telefonía.

GPL General Public License:
Licencia Pública General, es una licencia creada por la Free Software Foundation (FSF) y orientada principalmente a los términos de distribución, modificación y uso del software.

GSoC Google Summer of Code:
Es un programa anual que se dio por primera vez durante el verano de 2005 y que se ha ido repitiendo a partir de ese año. La empresa Google remuneró a los estudiantes que completaron un proyecto de programación de software libre durante ese periodo.

PDF Portable Document Format:
Formato de Documento Portátil, es un formato de almacenamiento de documentos.

GNU/Linux General Public License/Linux:
Es uno de los términos empleados para referirse a la combinación del núcleo o kernel libre similar a Unix denominado Linux, que es usado con herramientas de sistema GNU. Su desarrollo es uno de los ejemplos más prominentes de software libre; todo su código fuente puede ser utilizado, modificado y redistribuido libremente por cualquiera bajo los términos de la GPL.

IA Inteligencia Artificial:
Es la rama de la ciencia de la computación que se ocupa de la automatización de la conducta humana.

FP Falso Positivo:
Ocurre cuando un elemento es enmarcado dentro de una clase conocida y este no pertenece a dicha clase.

FN Falso Negativo:
Cuando el sistema devuelve que un elemento es desconocido y sin embargo pertenece una clase conocida.

Gb Gigabyte:
Es una unidad de almacenamiento de información cuyo símbolo es el Gb.

CISED Centro de Identificación y Seguridad Digital.

CEIGE Centro de Informatización de Entidades.

IDE Integrate Development Enviroment:
Entorno de Desarrollo Integrado. Herramienta que se utiliza para facilitar el desarrollo de software.

UML Unified Model Languaje:
Lenguaje Unificado de Construcción de Modelos.

CASE Computer Aided Sofware Engineerings:
Ingeniería de Software Asistida por Ordenador.

PCA Principal Component Analisis:
Análisis de los Componentes Principales.

LDA Linear Discriminant Analisis:
Análisis Discriminante Lineal.

ICA Independent Component Analisis:
Análisis de Componentes Independientes.

EBGM Elastic Bunch Graph Matching:
Correspondencia de Agrupaciones de Grafos Elásticos.

RNA Red Neuronal Artificial.

RBF Radial Basis Function:
Función de Base Radial.

RBFNN Radial Basis Function Neural Networks:
Red Neuronal de Función de Base Radial.

MLP Multi Layer Perceptron:
Perceptrón Multi Capa.

OpenCV Open Source Computer Vision:
Visión por Computadora de Código Abierto.

BSD Berkeley Software Distribution:
Distribución de Software de Berkeley.

GRASP General Responsibility Asignment Software Patterns:
Patrones Generales de Software para Asignar Responsabilidades.

GOF Gang of Four:
Banda de los Cuatro.

HU Historia de Usuario.

RGB Red, Green, Blue:
Rojo, Verde, Azul.

HSV Hue, Saturation, Value:
Matiz, Saturación, Valor. Es posible representar un color a través de la mezcla por adición de los tres colores primarios.

ORL Olivetti Research Laboratories :
Laboratorios de Investigación Olivetti.

TCP/IP Protocolo de control de transmisión/Protocolo de Internet:
Por sus siglas en inglés.

LTS Long Term Support:
Soporte a Largo Plazo.

PC Personal computer:
Computadora Personal.

RAM Random Access Memory:
Memoria de Acceso Aleatorio.

XML Extensible Markup Language:
Lenguaje de Marcas Extensible.

RF Requisitos funcionales.

RNF Requisitos no funcionales.

Glosario de términos

- ***Compactness:*** Medida que se utiliza para conocer cuán compacta es la forma de una figura.
- **Normalización:** Acción de transformar una distribución cualquiera a una distribución normal manteniendo la proporción de los datos.
- **Segmentación:** Proceso mediante el cual se divide una imagen en múltiples partes para lograr una mejor representación y obtener así información relevante.
- **Ecualización del histograma:** Técnica que consiste en ajustar los niveles de gris de una imagen para obtener una nueva imagen con un histograma uniforme.
- **Escala de grises:** Modelo de representación de una imagen digital mediante 256 tonalidades de gris.
- **Umbral:** Valor mínimo a partir del cual se considera que un elemento pertenece a una clase.