Universitat Politècnica de València

Descarga de computación de dispositivos móviles a ambientes Cloud Computing en un caso en concreto, el reconocimiento facial.

Tesina de Máster

Valencia, 15 de Junio de 2012

Autor:

Jorge Eloy Luzuriaga Quichimbo

Director:

Pietro Manzoni

A mis padres y hermanos con admiración y cariño.

Capítulo 1. Prólogo

Hoy en día todo el mundo tiene *cloud* y continuamente estamos oyendo *cloud, cloud, cloud* y si no es *cloud* es nube. *Cloud* tenemos de todos los proveedores para ver y tomar, *cloud* de servicio, *cloud* de plataforma, *cloud* de tal.. pero tenemos unas dudas: ¿el *cloud* es tan maravillo y genial como nos lo pintan? ¿el *cloud* es para todo el mundo?.

Hay aplicaciones que si que se pueden beneficiar de esta tecnología y aplicaciones que no la necesitan, cada aplicación es un caso, cada aplicación es un mundo aparte, para ello hay que evaluarlo.

Para involucrarnos, entender y participar en el desarrollo de esta tecnología la hemos evaluado con el reconocimiento facial desde dispositivos móviles, también comparamos ejecutar el mismo proceso de reconocimiento facial sin poseer la alternativa *cloud*. Y en base a esta comparación decidir que camino tomar...

Capítulo 2. Indice General

Capítulo 3. Introducción

Hoy en día tenemos toda una serie de dispositivos teléfonos móviles, PDA, tabletas, portátiles, maquinas de sobremesa pero si miramos mas allá ¿qué uso le damos a esta tecnología?, ¿en que invertimos la mayor parte del tiempo?, ¿estamos haciendo un procesamiento local o estamos accediendo mediante algún tipo de red al exterior? si lo pensamos bien vemos que en la mayor parte de los casos lo que estamos haciendo es acceder al exterior, comunicando de una manera u otra con otros computadores o con otras personas, siempre a través de una conexión en red.

La mayor parte del uso diario de los computadores no es como computador es decir como herramienta de cálculo sino como herramienta de comunicación a través de correo electrónico, foros, blogs, videoconferencias, incluso telefonía IP y por que no mencionar a los juegos multi jugador.

Si miramos desde un punto de vista más técnico implica que necesitamos toda una infraestructura para que toda esa comunicación se lleve a cabo y esta infraestructura es Internet. Internet tiene 2 grandes grupos de componentes por una parte la red de comunicaciones y por otra parte una serie de servidores que son los que procesan y almacenan la información que nosotros utilizamos.

De ahí que algunos nos hayamos centrado en las comunicaciones como tema de trabajo sobre el cual desarrollar nuestras inquietudes e intentar hacer que estas comunicaciones avancen lo más rápido posible con objeto de que puedan ofrecer un mejor servicio a la sociedad.

3.1. Objetivos

El objetivo de esta tesina es definir una aplicación para dispositivos móviles que nos sirva para medir las prestaciones de un entorno *cloud*, que pueda procesar datos en el dispositivo de forma local o realizar este procesamiento de forma remota en una entidad externa.

Mediante la ejecución de estos procesos en varios escenarios: el primero donde todos los procesos son ejecutados en el dispositivo móvil. Un segundo escenario donde el móvil simplemente sirve como terminal de entrada y salida de datos, y todo el procesamiento se realiza en remoto. Y un ultimo escenario basado en un modelo de colaboración entre procesos ejecutados en los dispositivos móviles y en el proveedor de *cloud computing*.

Finalmente buscar un equilibrio entre los diferentes tipos de procesos que pueda tener la aplicación y la distribución de los mismos para que se ejecuten entre el dispositivo móvil y el servidor *cloud*, por decirlo de una forma extrayendo lo mejor de ambos mundos.

Y por ultimo emular un canal inalámbrico entre el dispositivo móvil y el servidor *cloud* para ver bajo que limitaciones podemos obtener ventajas en el uso de esta técnica.

Capítulo 4. Antecedentes

4.1. Cloud computing

Los servidores tienen muchas utilidades pueden ser por ejemplo de búsqueda, servidores de webs, servidores de base de datos y otra tendencia que cada vez se está popularizando mas que son los centros de proceso de datos virtualizados en los cuales básicamente lo que se hace es sub contratar los servicios del centro de proceso de datos por empresas externas, por ejemplo una gran corporación bancaria tradicionalmente ha tenido sus propios centros de proceso de datos y sus respectivos expertos, pero ha llegado el momento en que estos servicios los están sub contratando, puede parecer erróneo pero es muy sencillo lo sub contratan porque reduce costes y a fin de cuentas les sale más barato,

¿Cómo es posible que algo sub contratado pueda salir más barato que si se lo hace en casa? Muy sencillo porque si tienen su centro de proceso de datos, no solamente tienen sus maquinas que necesitan mantenimiento sino que también tienen el consumo de energía eléctrica y el coste de mano de obra de los empleados que están trabajando en ese centro de cálculo.

Si hay varias empresas que sub contratan los servicios a la misma empresa, esta empresa puede comprar una maquina bastante más grande y potente, puede tener a menos personal que la suma de personal que tendrían todas estas corporaciones y también la factura de la electricidad es menor porque estas maquinas ahora pueden utilizarse de forma compartida en el tiempo. El punto clave aquí es la virtualización, el poder partir una maquina física en múltiples maquinas virtuales cada una de ellas ejecutando su propia copia de sistema operativo y aunque tengamos la misma máquina física hay unos niveles de seguridad de protección que hacen que empresas como los bancos, aseguradoras, etc. confíen sus servicios a estas empresas.

Hay algunas empresas que han dicho sería mejor negocio que si en lugar de dar este servicio a una corporación bancaria o una empresa de seguros, lo damos a todos los usuarios de internet del mundo, así contamos con una clientela muchísimo más amplia, y entonces lanzan el *Cloud Computing*.

4.2. Cloud Computing la idea

La idea es: usted ya no necesita siquiera un portátil para procesar localmente lo que quiera, nosotros le hacemos el procesamiento local, lo haremos en nuestros servidores usted simplemente va a necesitar un equipo con muy poquita potencia que tenga conexión a red y el navegador estándar, así igual que navega por internet accede a cualquier tipo de aplicación que nosotros ofrecemos y por otra parte en lugar de pagar los elevados precios de las licencias de las aplicaciones que posiblemente apenas utilice, se paga por servicios igual que pagamos con los teléfonos móviles "tanto usas tanto pagas" es decir una aplicación en concreto se ejecuta de forma remota y solamente se paga si la hemos usado, que puede salir a cuenta si la hemos usado pocas veces, esta es la idea de negocio que hay detrás de la iniciativa del *Cloud Computing*.

Por ejemplo un usuario dice: yo necesito ahora utilizar una hoja de cálculo: se conecta al servidor

utiliza la hoja de cálculo, transfiere los datos, y termina.

De esa forma, hay empresas que han instalado enormes servidores de estas aplicaciones y de esta forma proporcionan servicios de software como servicio *SaaS*, plataformas como servicio *PaaS*, e incluso infraestructuras como servicio *IaaS*, llegando al punto de ya no es solo una aplicación sino un computador personal completo.

Si bien para ofrecerle al ciudadano en la calle todavía es un poco pronto porque hay grandes temores de ¿qué hago yo por ejemplo si falla la red? ó ¿mis datos privados están seguros? Pero para una empresa que tenga múltiples sedes puede ser bastante interesante, por ejemplo una empresa que tiene muchas sedes y su personal comercial tiene que moverse de un sitio para otro para visitar a clientes, hoy están en una oficina aquí y mañana están en otra. En lugar de tener que llevar su portátil siempre encima con todos sus datos, documentos, etc., el que puedan tener un servidor con un PC virtual dedicado y ejecutando las aplicaciones que necesitan así en un momento determinado el comerciante tiene que irse, cierra el terminal y el PC virtual sigue ejecutándose en el servidor, entra en un modo de bajo consumo para no consumir demasiado y cuando llega a otro sitio o bien desde el aeropuerto se conecta al PC que sigue estando exactamente en el mismo punto en que estaba y continua su trabajo desde otro lugar.

Como planteamiento para empresas con múltiples sedes no es una mala idea y permitiría a largo plazo reducir costes.

Otro ejemplo en el que podemos ver los beneficios de CC es una empresa donde una persona que quería disponer de un ordenador para probar la funcionalidad de algún software que está creando o instalar un servidor, tenía que ir a hablar con el administrador de sistemas pedirle que por favor monte una maquina de tales características, esperar a que hubiese esa máquina disponible o que la compraran y en muchos de los casos no se podía seguir trabajando hasta que esa solicitud pase la aprobación de varias personas. Con el tema de la infraestructura como servicio podemos hacer todo esto en la nube diciendo solamente “quiero una maquina” a una página web y en 4 minutos la tenemos con las características que la hemos pedido.

Todo el proceso de creación de las maquinas virtuales esta automatizado por un software que es capaz de preparar toda un infraestructura nosotros simplemente decimos que queremos tal sistema operativo, que queremos que tenga tanta memoria, que tenga tanto poder de cálculo, que tenga red, etc.

Basándose en este software hay empresas que tienen muchos ordenadores en su centro de computo y no los están utilizando 24 horas del día 7 días a la semana, así que prefieren empezar a vender su poder de cálculo a otras empresas que necesitan estos recursos de forma esporádica para una situación especifica por ejemplo cubrir eventos como elecciones, foros, convenciones, etc., este es un ejemplo de cómo empezó *Elastic Computing Cloud* EC2 de Amazon.

4.3. Promotores de Cloud Computing

Muchas personas están familiarizadas con el termino *Cloud Computing* gracias al éxito de empresas como Amazon y SalesForce.com, también a aplicaciones como Google Docs, **iCloud** de Apple que es una solución basada en almacenamiento en la nube para e-mails, notas, libro de contactos, fotos y documentos con el que todos los usuarios pueden automáticamente sincronizar sus iMac, iPod, iPhone entre otros dispositivos Apple. Microsoft por su parte tiene **LiveMesh** que integra ordenadores con sistema operativo Windows, teléfonos con Windows Mobile, Xbox, e incluso ordenadores Mac interconectando estos dispositivos a través de internet permitiendo así a los usuarios sincronizar los datos de todos estos dispositivos por internet y almacenarlos en la nube.

4.4. Mobile Cloud Computing - Nube móvil

Sin embargo un menor número de personas conocen Mobile Cloud Computing MCC pero se puede deducir muy fácil desde el concepto de CC que es una arquitectura de computación proporcionada por un centro de proceso de datos que en función de las necesidades de los usuarios es capaz de proveer servicios bajo demanda de forma flexible y automatizada, estos servicios pueden ser aplicaciones, infraestructura o plataforma, MMC comparte la misma idea con la distinción que estos servicios son únicamente accedidos desde plataformas móviles y a los que estos servicios deben de alguna manera ofrecer ahorro de energía.

MCC fue definida como la disponibilidad de servicios cloud en un ecosistema móvil, en marzo del 2010 en [15].

Se puede utilizar los términos Wireless o Mobile indistintamente pero se prefiere Mobile por referirse a cualquier lugar y en cualquier momento y no Wireless que significa sin cables, por lo tanto cuando hablamos de MCC nos referimos a acceso seguro a nuestros datos en cualquier lugar y en cualquier momento.

El ecosistema móvil engloba a clientes, empresas, dispositivos móviles, femtoceldas, transcodificación, seguridad de extremo a extremo, puertas de enlace y servicios habilitados de banda ancha móvil.

4.5. Las Plataformas

Entre los dispositivos móviles como pueden ser teléfonos inteligentes, tabletas, etc. La plataforma más ubicua son los teléfonos móviles, seguidos por las tabletas que están comenzando a impregnarse en el mercado, hay muchos fabricantes de estos dispositivos que los podemos agrupar por los sistemas operativos más usados que son: Android, iOS, Windows Mobile, Symbian y RIM BlackBerry, algunas de sus principales características podemos observar en la *tabla 1.*

Nombre S.O.	Desarrollado por	Filosofía	Recursos para desarrolladores	Mercado de distribución de aplicaciones
BlackBerry	RIM - Research in Motion	Propietaria	Java con BB. IDE + simulador	BB. AppWorld MobiHand
Android	Google	OpenSource	Java con Android SDK + emulador	Google Play GetJar Handango
iOS	Apple	Propietaria	Objective-c Apple iOS SDK xCode Interface Builder + simulador	Apple App Store
Windows Phone 7	Microsoft	Propietaria	Visual Studio con APIs + simulador	W.P. Market Place
Symbian	Nokia	OpenSource y capas Propietarias	Java ME C++ Perl + simulador	Ovi Store

Tabla 1. Principales plataformas de dispositivos móviles inteligentes.

Las tabletas poseen un tamaño superior al de un teléfono móvil pero interactúan con el usuario de la misma forma usando la pantalla táctil como principal dispositivo de entrada, el top de ventas se lo lleva Apple con el iPad.

La gente cada vez está reemplazando sus viejos terminales por teléfonos inteligentes por lo cual se tiene un nicho de mercado mundial para el desarrollo de aplicaciones móviles.

Estos dispositivos han redefiniendo la manera en que los usuarios consumen los contenidos, antes llegábamos a casa y para poder navegar íbamos a nuestro ordenador y abríamos el navegador, si queríamos consultar las noticias íbamos a la página web de nuestro periódico favorito, si queríamos saber los resultados de nuestro equipo hacíamos lo propio, todo esto está cambiando gracias a la inclusión de los móviles y lo que son las aplicaciones. Ahora los usuarios tienen una aplicación para cada uno de sus temas de consumo: una aplicación para consultar los resultados deportivos, otra para el tema de las noticias, una para la mensajería instantánea, etc.

El lugar también está cambiando los usuarios consumen contenidos en las calles, en los autobuses en las pausas entre clases, esto es el paradigma móvil que está haciendo que el consumo de los contenidos se dirija a un consumo mucho mas rápido, agresivo y de calidad.

Los hábitos de trabajo también están cambiando, los expertos sugieren que para el 2020 la mayoría de las personas que usan internet trabajaran a través del ciber espacio lo cual incrementa el riesgo de seguridad, sin embargo como una política de seguridad de acceso a los datos se considera el uso de la geolocalización. Por otra parte IBM pronostica que para el 2015 habrá un trillón de dispositivos preparados para la nube en manos de la gente no tanto para explotar la información sino para generarla.

4.6. Problemas

Un problema notable son los recursos limitados que actualmente encontramos en los dispositivos móviles, si los comparamos con los equipos de sobremesa, estos tienen menos pantalla, menos memoria, menos potencia de cálculo y la energía de la batería está limitada a unas cuantas horas de uso. Por estas limitaciones MCC es visto como SaaS que significa que la computación y el manejo de los datos son realizados y alojados en la nube y accedidos a través de un navegador de internet o una aplicación cliente.

La latencia y el ancho de banda también afectan al MCC, si bien WiFi mejora la latencia pero el ancho de banda se decrementa si muchos dispositivos móviles están conectados al vez. En la red celular 3G el ancho de banda en muchas zonas está limitado por el ancho de banda de la antena de telefonía, además la conectividad es intermitente e irregular fuera de las zonas urbanas, lugares como el subsuelo de un edificio, el interior de un túnel y pasos subterráneos.

Por último tenemos los problemas de seguridad que también se incrementan con los dispositivos móviles, en el lado del cliente estos se pueden perder con mayor facilidad y todos los datos sensibles almacenados en el mismo están a merced de cualquier persona malintencionada. En el lado del servidor como los datos están almacenados y manejados en el cloud la seguridad y la privacidad de los mismos depende del proveedor de este servicio, esto puede plantear graves preocupaciones a empresas que almacenen sus secretos comerciales o a usuarios que almacenen sus ideas patentables. Es por esto que muchos prefieren tener los datos sensibles bajo su control para asegurar su privacidad y seguridad.

4.7. Soluciones Tecnológicas

Para el grupo de **problemas** relacionados con la **falta de recursos** se han desarrollado diferentes tecnologías que intentan resolver el problema, entre ellas tenemos:

4.7.1. CloudClone

[1] La idea es clonar completamente los datos y las aplicaciones del dispositivo móvil y ubicarlas en los servidores *cloud*, de esta manera cuando se ejecuta alguna operación se realiza selectivamente si es sencilla en el dispositivo móvil o si es compleja en el clon y se sincroniza el resultado.

4.7.2. AlfredO

[2] A través de Alfredo, las capacidades de un teléfono representan elementos modulares de servicio que se pueden acceder sobre la marcha por cualquier teléfono móvil.

AlfredO es una arquitectura *middleware* que permite desarrollar aplicaciones basadas en un modelo de dependencias de manera modular organizando en capas desmontables que se pueden distribuir para configurar dinámicamente arquitecturas de múltiples niveles entre los teléfonos móviles y proveedores de servicios. Haciendo del teléfono móvil un punto de acceso universal a cualquier infraestructura.

4.7.3. Cloudlet (la nubecilla)

[3] Con el fin de proveer una solución para situaciones donde la latencia es muy alta se acerca los recursos de *clouds* distantes a los dispositivos móviles, obteniendo así pocos usuarios conectados al mismo tiempo a la nubecilla frente a los cientos o a miles que pueden estar conectados a la nube además las latencias y anchos de banda son los de una red local y no la latencias de acceder por internet a la nube.

4.7.4. Hyrax

[4] Hyrax es un prototipo basado en ideas de CC para construir una plataforma MCC usando teléfonos móviles y servidores formando así dos tipos de redes una de teléfonos inteligentes y otra que sería una red heterogénea de teléfonos y servidores, todo esto con el fin de hacer frente a las limitaciones de espacio de almacenamiento, a la capacidad de cómputo y a los cuellos de botella de la red en un prototipo que escala con el número de dispositivos.

Para los **problemas de latencia y ancho de banda**, tenemos soluciones como:

4.7.5. 4G

Es la evolución de la velocidad de navegación 3G es decir conseguir una velocidad similar a lo que se está ofreciendo con las redes de fibra óptica de hoy en día 50 y 100 megas de navegación incluso a alta velocidad, por ejemplo si alguna persona trabaja con su móvil y viaja en tren se debe encontrar a menudo que navegar a grandes velocidades es complicado, 4G viene a solucionar esta parte y viene a dar mejor cobertura.

Además unificara las conexiones móviles con las conexiones de ordenadores estándar obteniendo una sola red para todos los dispositivos.

4.7.6. HTML5

Con las nuevas especificaciones del estándar las aplicaciones web para móviles se benefician por las mejoras que se introduce, recodemos que HTML es un lenguaje de marcado de publicación de documentos que proporciona un medio de especificar elementos de páginas web tales como títulos, textos, tablas, listas y fotos.

Entre las mejoras que tenemos esta el soporte *offline* que hace posible el almacenamiento local de datos como el e-mail o el calendario que ayuda a las interrupciones de conectividad, otra es que agrega el objeto *canvas* para gráficos y funciones de video evitando la necesidad de instalar plugins como Flash de Adobe o Silverlight de Microsoft. Además cuenta con un API de geolocalización.

HTML5 trabaja con CSS3 para especificar cómo deben ser renderizados los elementos de la página, HTML dice que mostrar y CSS dice como mostrarlo.

Frente a los **problemas de seguridad** la virtualización es fundamental junto con unos niveles de seguridad y privacidad aun así hay información que evidentemente no puede ser almacenada en la nube, la solución existente es encriptación de los datos antes de almacenarlos, esto previene accesos desautorizados en caso de fallos o agujeros del proveedor *cloud*. También está el uso de nubes híbridas donde las empresas almacenan ciertos recursos en casa especialmente sus datos y los servicios son sub contratados a terceros.

Capítulo 5. Análisis

Actualmente el poder de computo y el almacenamiento necesario de los dispositivos móviles no esta diseñado para satisfacer las aplicaciones más exigentes, para solventar este problema este tipo de aplicaciones puede adquirir recursos de la nube.

En este capítulo nos encontramos con la cuestión de qué tipo de aplicación nos puede servir para medir las prestaciones del uso de la nube.

Necesitamos una aplicación cliente móvil que tome la forma de una aplicación Android que tenga una doble función: la de captura y la de visualización *front-end* de los resultados generados por el *back-end* alojado en la nube.

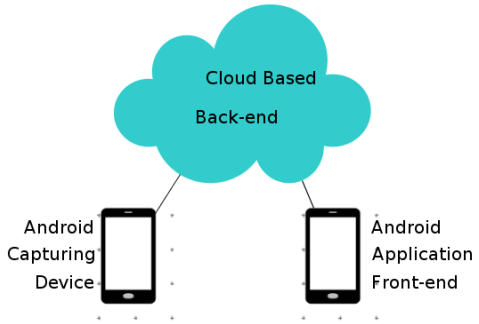


Figura 5.1. Interacción dispositivos móviles en un entorno cloud computing

Para Kumar [5] los procesos que requieren poco poder de cómputo son realizados en el dispositivo móvil y los procesos que requieren de un considerable poder de computo se realizan en los servidores de los centros de proceso de datos de las grandes compañías, una especie de delegación de tareas, un outsourcing asistido por la nube.

5.1. Outsourcing de procesos

Debemos buscar un equilibro entre los diferentes tipos de procesos que pueda hacer una aplicación, como se muestra en la *figura 5.2* para obtener el resultado *r* con las entradas *x,y,z* se divide la aplicación a groso modo en cuatro módulos principales *A, B, C, D* donde *A y B* por ser procesos sencillos se realizan en el móvil y los procesos C y D son procesos complejos que necesitan computo intensivo.

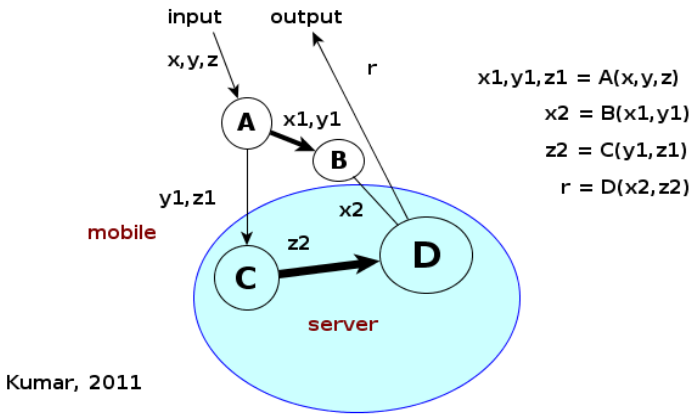


Figura 5.2. Particionado de una aplicación para distintos entornos

La descarga de procesos hacia la nube es beneficiosa cuando se requiere grandes cantidades de poder de cómputo con pequeñas cantidades de comunicación o transferencia de información, y las combinaciones intermedias entre poder de computo y comunicación pueden ser beneficiosas o no dependiendo del ancho de banda del canal, como lo podemos observar en la *figura 5.3*

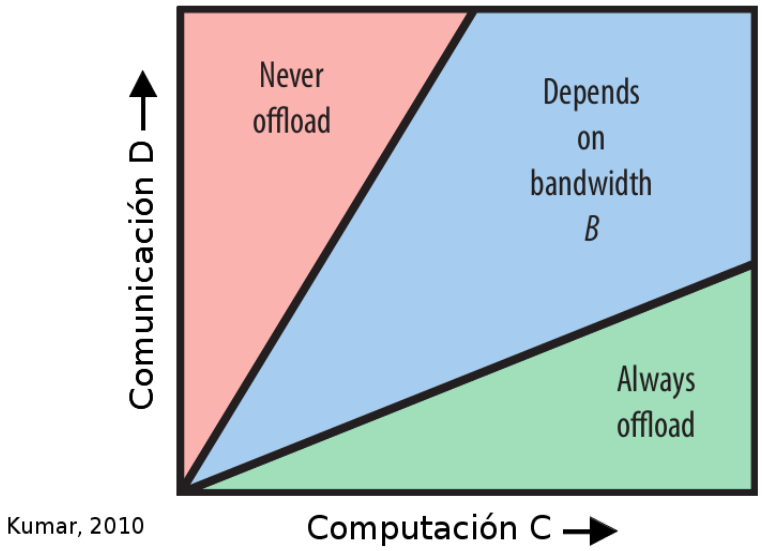


Figura 5.3. Cuando resulta beneficioso descargar poder de computo

Siguiendo el diseño tradicional cliente-servidor o el de aplicaciones web, la aplicación se divide o se particiona para que se pueda ejecutar parcialmente en el dispositivo y parcialmente en la nube, para dividir las aplicaciones en componentes modulares se debe tener muy en cuenta: la funcionalidad, la demanda de cómputo, las dependencias de datos, y las necesidades de comunicación.

Además este particionado debe cumplir dos requisitos fundamentales: el primero es que la ubicación o ambiente donde se ejecuta el componente no afecta su funcionalidad y el segundo es que la ubicación de los componentes debe ser transparente al usuario.

Por otra parte la configuración de la aplicación se determinada cuando se lanza a ejecución, es decir la aplicación se configura dinámicamente asignando módulos a distintas unidades de ejecución tanto en el dispositivo móvil como en la nube. Y pueden estar replicados así el fallo en una instancia no afecta al resultado de la aplicación.

Uno de los beneficios que salen a la vista son: que los dispositivos móviles no están limitados por las capacidades de cómputo ni por el espacio de almacenamiento de su hardware, si lo requieren estos se obtienen de la nube.

Desde el punto de vista del rendimiento, este se incrementa y se optimiza para ajustarse a objetivos como: sensibilidad, coste y consumo de energía, factores que prolongarían el tiempo de vida de las aplicaciones.

Consumo de energía

Cada aplicación ejecutándose en un dispositivo móvil consume energía por el uso del CPU, la memoria y los módulos de radio comunicación. Una aplicación dividida en componentes depende principalmente de la ejecución de operaciones de entrada/salida y de la comunicación a través de los diferentes canales. Lanzar a ejecución un modulo podría idealmente ahorrar el consumo de energía en el dispositivos debido a evitar realizar el computo pero este beneficio puede ser anulado por el consumo de energía de las interfaces de red.

Un estudio amplio sobre el consumo de energía en los dispositivos móviles podemos encontrar en[16] uno de los casos más dramáticos es el consumo del sensor GPS, como lo podemos observar en la *figura 5.4* que en hora de funcionamiento ha consumido más del 20 por cien de la batería.

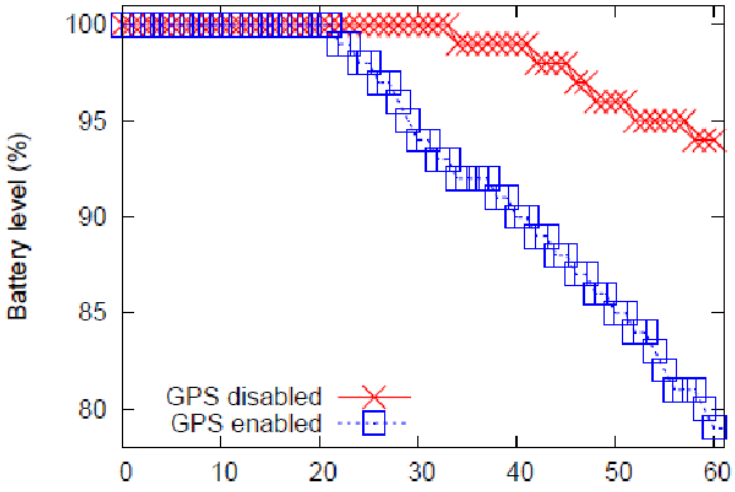


Figura 5.4. Consumo del sensor GPS en una hora de uso

Así como el GPS otros sensores continúan consumiendo la misma cantidad de energía por un periodo de tiempo considerable como cuando estaban en funcionamiento después de haber sido desactivados, otro de los aspectos se tiene muy en cuenta es el consumo apertura y lectura de ficheros

5.2. Aplicaciones

Cada vez vemos mas aplicaciones y juegos que utilizan este tipo de tecnología, entre los tipos de aplicaciones computacionalmente intensivas tenemos: procesamiento de medios digitales, minería de grandes cantidades de datos, visión y reconocimiento, juegos, procesamiento de textos, como lo vemos en la *figura 5.5* donde además nos indica que la plataforma favorita por los desarrolladores son los teléfonos inteligentes.

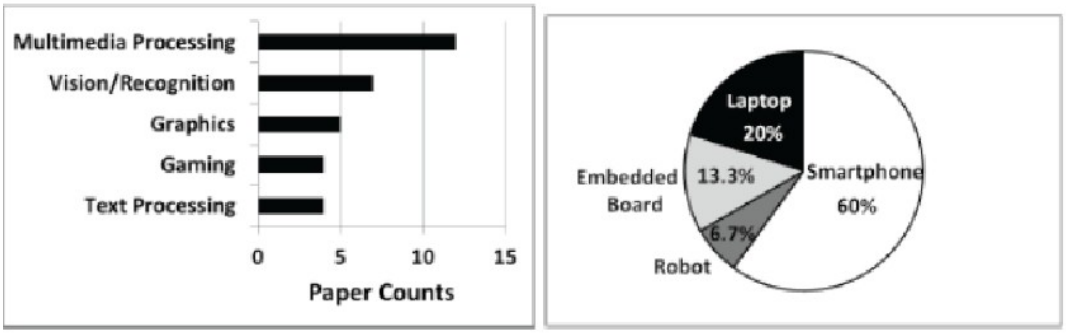


Figura 5.5. Tipos de aplicaciones y plataformas preferidas en los artículos.

Entre algunos ejemplos más específicos de estas aplicaciones nos podemos encontrar hoy en día con: portales móviles e-gobierno, aplicaciones para encontrar el nombre de personas con una foto tomada en la calle, e incluso procesamiento de movimientos inusuales capturando con la cámara enviando varios *frames* por segundo, todas estas APPS con una característica en común que es la captura, extracción de características como datos *input* enviados a la nube para su procesamiento y respuesta con los resultados de *output*.

Por nuestra parte necesitamos una aplicación puedan ser ejecutada tanto en el sistema móvil como en un servidor en la nube, que en base modificaciones a los valores del comportamiento canal, anchos de banda, latencias y tasas de errores nos conteste a la pregunta ¿para qué valores es razonable hacer computo local? y ¿para qué valores hacer computo remoto?.

Partiendo de los ejemplos planteados anteriormente elegimos el desarrollo y pruebas de una aplicación similar a la de que en base a una imagen reconocer a la persona que aparece en la misma debido a que es un campo tecnológico muy amplio con múltiples aplicaciones industriales y comerciales, lo hacemos con personas pero esta misma arquitectura y metodología se podría aplicar por ejemplo para reconocimiento de sonidos, objetos, coches o situaciones especiales de tráfico, pero claro aquí el factor más importante es la latencia que consigamos para podernos extender en ese sentido.

La elección de este tipo de aplicación es un verdadero reto porque conlleva una conjunción de diferentes áreas y tecnologías como el procesamiento de imágenes e inteligencia artificial en ambientes de cómputo heterogéneo además del manejo y dependencias de datos para las comunicaciones, etc.

5.3. Reconocimiento automático de rostros

Los seres humanos desde los primeros días de vida estamos listos para reconocer rostros, es una habilidad natural, sin embargo desde de los años 60 se ha planteado hacerlo de forma automática ya en los años 90 aparecen técnicas que se han ido perfeccionando progresivamente con el paso del tiempo llegando hasta nuestros días donde incluso se posee información 3D del rostro.

El reconocimiento facial de forma automática es una de las técnicas biométricas que permite reconocer personas a partir de rasgos distintivos extraídos de una imagen. Este proceso fluye en cuatro etapas: captura de muestras, extracción de características, comparación en plantillas y *matching* (emparejamiento).

Dependiendo de la forma con la que se extrae las características existen varios métodos para reconocer personas, uno de los mas utilizados actualmente es Eigenfaces que fue desarrollado en el *Massachusetts Institute of Technology* MIT, quienes motivados por representar eficientemente rostros usando el análisis de componentes principales PCA han alcanzado caracterizar un conjunto de caras con un mínimo numero de parámetros, en [9] hacen una interesante revisión a otras técnicas.

El PCA se basa en las propiedades estadísticas de las imágenes y es un método óptimo para reducir el número de dimensiones necesarias para representar un conjunto de vectores.
Consideremos por ejemplo un conjunto de las imágenes de rostros, los rostros son relativamente semejantes y todos tienen los mismos elementos situados de forma semejante por lo tanto este conjunto de las imágenes de rostros se podría representar en un espacio de menor dimensionalidad.

Una abstracción de estos conceptos los podemos ver en la *figura 5.6* sobre la Mona Lisa se han dibujado dos flechas la roja que va desde el pecho hacia el rostro y la azul que va desde el pecho hacia el hombro ambas flechas representan vectores que contienen valores característicos de esta imagen, después de una transformación la Mona Lisa ha sido deformada y la flecha azul ha cambiado de dirección mientras que la roja se mantiene intacta así el vector representado por la flecha roja es el vector propio o vector eigen de esta imagen, entonces después de aplicar PCA a este vector obtenemos un vector ortonormal donde la suma de todos sus valores o valores eigen es igual a 1.

La verdadera reducción de dimensionalidad de la representación mediante eigenfaces consiste en utilizar solo las componentes con mayor valor propio [6].

Utilizaremos eigenfaces en nuestra aplicación para el reconocimiento de rostros, así cuando busquemos un rostro dentro del conjunto de imágenes de entrenamiento el sistema hace el *matching* entre el rostro buscado con la imagen alguna de las imágenes de entrenamiento de la misma persona.

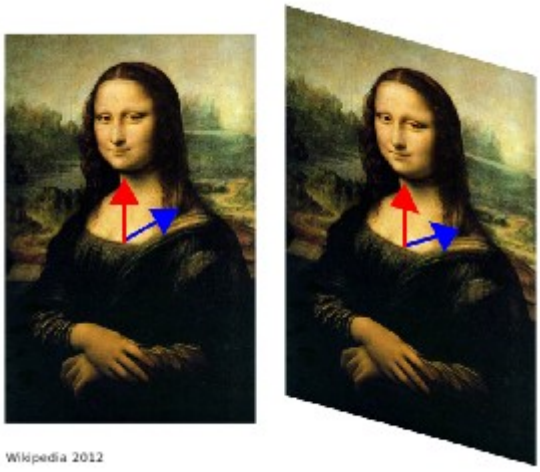


Figura 5.6. Mona Lisa representada con eigenVectores

5.4. Servicios web de Amazon

Nos centramos en el enfoque de que los sistemas móviles no realizan la computación, la computación se realiza en algún otro lugar, enviar computación a otra maquina no es una nueva idea es la misma del súper conocido modelo de computación cliente-servidor, uno de los pioneros en Cloud Computing que mas solida tiene esta tecnología es Amazon mediante sus Amazon Web Services.

Amazon Web Services AWS ofrece a empresas de todos los tamaños y de todos los rincones del mundo una plataforma informática en la nube que es flexible, escalable y de bajo coste, con el objetivo de que sus clientes empleen el tiempo en centrarse en su idea de negocio y no en aprovisionar y ni gestionar hardware.

Es flexible porque permite que el cliente sea quien elija sistemas operativos, lenguajes de programación, herramientas web para ejecutar e implementar sus aplicaciones. Es escalable por que se tiene acceso a la enorme infraestructura de Amazon y en función de la demanda se puede utilizar los recursos de forma fiable y segura. Y de bajo coste porque se basa en la filosofía de pago por uso donde se paga por los recursos que utilizan, no existen contratos a largo plazo y ni de compromiso de permanencia.

Además pone a disposición del cliente la consola de administración web, un conjunto de APIs para varios lenguajes de programación, SDKs y otras herramientas de desarrollo junto a guías de introducción, tutoriales, foros de la comunidad para empezar a utilizar AWS en cuestión de minutos.

Otra ventaja para los nuevos clientes es que nos permite utilizar AWS de forma gratuita durante un año o 750 horas de uso con la posibilidad de ejecutar instancias[1] de *Amazon Elastic Computing*

1 Amazon utiliza el término instancia para referirse a lo que nosotros conocemos como maquina virtual que puede ser un servidor o un ordenador.

Cloud EC2 junto a otros potentes servicios complementarios de alta disponibilidad como espacio de almacenamiento S3, balanceo de carga *Elastic Load Balancing,* backups automáticos de datos, distribución de contenidos con muy baja latencia *CloudFront,* alquilar discos de alta disponibilidad para las instancias *EBS,* bases de datos relacionales RDS, crear o destruir instancias dependiendo de la demanda que tenemos *Auto Scaling*, servidores DNS distribuidos *Route* 53 y el SES que es de servicio de correo electrónico, entre otros que los podemos ver en la *figura 5.7.*

Figura 5.7. Listado de servicios a los que podemos acceder con los servicios web de Amazon.

La verdad que con tantos servicios que tenemos a disposición ejecutando y probando las 750 horas gratuitas nos quedan cortas, cuando hemos superado el límite de uso gratuito, comenzamos a pagar las tarifas de servicio estándar según el uso.

5.5. Reconocimiento facial por face.com

La primera opción era implementar la aplicación de reconocimiento facial en el servidor por nuestra cuenta y subirlo a una instancia en los servicios web de Amazon pero en la revisión de la literatura nos encontramos que actualmente existen varias empresas dedicadas cien por cien a estas actividades, especializadas en el tema que ofrecen el Reconocimiento como Servicio siguiendo la filosofía del *Cloud Computing*, de las cuales podemos destacar a *bioID* y a *face.com* ambas ofrecen una API para poder interactuar con su plataforma desde cualquier otro terminal con conexión a Internet, pero con la única diferencia de que *face.com* ofrece sus servicios de manera gratuita, por lo cual sin mas empleamos esta tecnología para nuestro estudio.

Face.com es una empresa tecnológica con el mejor software de reconocimiento facial en su clase que ofrece una plataforma robusta y gratuita que automáticamente detecta y reconoce rostros en cualquier imagen.

Gracias a la uso de la plataforma por parte de usuarios y desarrolladores han logrado indexar casi 31 billones de imágenes de rostros de más de 100 millones de individuos con un nivel de confianza media de 91.3% ± 0.3. [11]

Su gran éxito en el reconocimiento se debe a la técnica de reconstrucción del rostro en un espacio 3D con la que trabajan la cual es suficientemente robusta para enfrentar retos como manejar y tratar con imágenes de baja resolución, variaciones en poses, iluminación y sobre exposición. Haciendo que estas pasen por un proceso de normalización en el cual se transforman a vistas frontales utilizando y calculando simetrías existentes en el rostro y proyectándolas en espacios 3D, como podemos ver en la *figura 5.8*

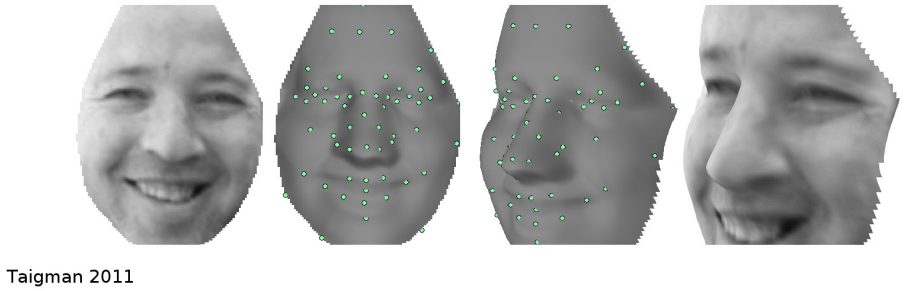


Figura 5.8. identificacion de las principales caracteristicas faciales y reconstrucciones en 3D utilizadas en face.com

Sus algoritmos permiten procesar en tiempo real la detección y reconocimiento de rostros en mas de hasta 30 *frames* por segundo que son reconstruidos en precisos modelos 3D a partir de una sola imagen, Además usando modelos discriminatorios basados en el orden de billones de rostros pueden realizar la estimación de la edad y reconocimiento de expresiones faciales o estado de ánimo, todo esta complejidad actualmente se ejecuta sobre servidores con arquitectura *Sandy Bridge* de Intel con 8 cores. [11]

Toda esta infraestructura y tecnología envuelta en el reconocimiento facial esta disponible por medio de una API REST y depende de la creatividad de los desarrolladores para crear aplicaciones originales.
Así por ejemplo nosotros lo único que le tenemos que pasar es una imagen que contenga el rostro que queremos reconocer y este nos devuelve el resultado en forma de un mensaje en XML o JSON.
Para empezar a usar esta API en la misma página web encontramos herramientas, ejemplos y documentación para familiarizarnos y poder explorar las funciones y respuestas.

Para promover uso saludable y disuadir el *spam* los servicios de *face.com* están sujetos a un máximo de cinco mil consultas en una hora y a la siguiente hora se reinicia el contador, lo que no significa ninguna limitación incluso para aplicaciones en tiempo real que envíen hasta una imagen por segundo.

5.6. Libreria OpenCV

OpenCV significa *Open Source Computer Vision Library* creada por Intel, es una colección de funciones en lenguaje C y clases en C++ que implementa los más populares algoritmos de procesamiento de imágenes y visión por computador para aplicaciones que incluyan análisis de movimientos y seguimiento de objetos, análisis de imágenes, análisis estructurales, reconocimiento de objetos, reconstrucciones 3D, etc.

OpenCV es una librería de código abierto, gratuita, multiplataforma que esta continuo desarrollo. Para el dominio específico de los rostros contiene funcionalidades avanzadas para la detección y seguimiento del rostros mediante algoritmos AdaBoost y para el reconocimiento mediante objetos eigen.

El algoritmo AdaBoost busca una combinación optima de un conjunto de clasificadores débiles con el fin de crear uno más potente dentro de un proceso iterativo, los clasificadores débiles se obtienen mediante la aplicación de *wavelets* y un operador definido en una región rectangular, donde una parte de los pixels se suman o se restan y si el valor del filtro está por encima de cierto umbral se acepta o se rechaza, esta es la propuesta de Viola [7]. Esta clasificación basada en filtros de Haar y AdaBoost fue implementada en OpenCV por Lienhar [8].

La decisión en la clasificación de que un objeto en una imagen es o no un rostro depende si estos objetos han pasado por todos los filtros alrededor de 30 ejecutados en cascada y de forma secuencial, como podemos observar en la *figura 5.9*.

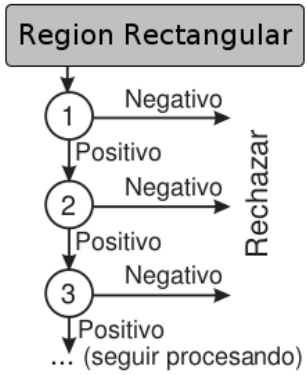


Figura 5.9. Esquema del clasificador en cascada de rostros

La implementación de los clasificadores en OpenCV es genérica y entrenable e incluye cascadas ya entrenadas como ficheros XML y llevan como prefijo “haarcascade_”, para encontrar el cuerpo de una persona, rostros de frente y de perfil y otras para encontrar regiones especificas del rostro como ojos, nariz, boca, orejas, como lo vemos en la *tabla 2.*

Clasificador	Para detectar
frontalface_alt	La cara frontal
frontalface_alt2	La cara frontal
frontalface_alt_tree	La cara frontal
frontalface_default	La cara frontal
profileface	La cara de perfil
fullbody	Todo el cuerpo
lowerbody	Parte inferior del cuerpo
upperbody	Parte superior del cuerpo
mcs_upperbody	Parte superior del cuerpo
eye	Ojo
mcs_eyepair_big	Ojo
mcs_eyepair_small	Ojo
eye_tree_eyeglasses	Gafas
lefteye_2splits	El ojo izquierdo
righteye_2splits	El ojo derecho
mcs_lefteye	El ojo izquierdo
mcs_righteye	El ojo derecho
mcs_mouth	Boca
mcs_nose	Nariz
mcs_leftear	Oreja izquierda
mcs_rightear	Oreja derecha

Tabla 2. Clasificadores estándar en OpenCV

Nos apoyamos en esta librería de procesamiento de imágenes para el desarrollo de nuestra aplicación.

5.7. Desarrollo de aplicaciones con Android

El lenguaje en el que se desarrolla aplicaciones para la plataforma Android es Java.

5.7.1. Herramientas

Es necesario tener un entorno de desarrollo IDE, nosotros usamos Eclipse que es el que Google recomienda para desarrollar aplicaciones para su plataforma móvil, le añadimos el *plugin* que nos proporciona la integración del SDK de Android con este entorno de desarrollo y con estas herramientas en mano sin la necesidad de poseer siquiera de un teléfono Android porque incluye emuladores y sistemas de logs comenzamos a trabajar con las potentes funcionalidades del sistema.

Para desarrollar aplicaciones sencillas es suficiente con el emulador, un dispositivo real lo necesitamos cuando vamos a interactuar con los sensores y la cámara.

5.7.2. Entidades

La principal entidad con la que trabajamos son las *activities* que no son mas que las pantallas de la aplicación. A cada *activity* se le puede asociar un fichero XML que corresponde al *layout* de la vista que va a ver usuario, muy parecido a HTML en la creación de páginas web de forma que podemos separar lo que es la lógica de lo que es la visualización.

Por otro lado tenemos los servicios que no tienen ningún tipo de vista y se ejecutan en *background* o segundo plano. Los servicios y las *activities* se pueden comunicar para intercambiar información entre si mediante *intents*.

Otro tipo de proceso que se ejecuta en segundo plano en un breve espacio de tiempo son las tareas asíncronas que permiten seguir ejecutando otros procesos sin bloquear el hilo principal, por ejemplo la lectura o escritura de la base de datos, conectarse a un servidor remoto, etc. Android tiene un tiempo de espera máximo que está fijado en 5 segundos y si no se concreta los resultados en este tiempo aparece un mensaje que dice que el proceso está tardando demasiado y nos pregunta si lo queremos terminar o seguir esperando.

Para escuchar eventos que ocurren en el exterior Android tiene los *broadcast receive* que son procesos que están esperando el momento que ocurra cualquier evento para ejecutar una o varias tareas.

5.7.3. Vistas

En el momento en que giramos la pantalla del móvil para todas las aplicaciones, se pueden utilizar dos tipos de *layouts,* el modo *portrait* que es el modo vertical y el modo *landscape* que es el modo horizontal, estos se almacenan en la carpeta *layout* dentro de la carpeta de recursos, donde además tenemos tres carpetas con el prefijo *drawable* que contienen las imágenes a utilizar por una aplicación según si la densidad de pixels de la pantalla en la que ejecutemos la aplicación es baja, media o alta, de este modo cuando estamos viendo en una pequeña pantalla en modo horizontal, Android va a la carpeta de *layout* y coge el *layout* en horizontal con las imágenes con baja densidad de pixels de forma que lo podamos ver bien, si giramos el teléfono nuevamente y lo ponemos en vertical Android recarga la actividad, va a la carpeta de *layout* en modo *portrait* y coge el layout e imágenes correspondientes.

Podemos utilizar otros componentes Android a nuestro gusto dependiendo de lo que necesitemos de la aplicación, definiendo incluso elementos que luego se puedan reciclar entre aplicaciones.

5.7.4. Gestión de los recursos

Todo queda centralizado en una clase que se denomina R.java que es un sitio en el cual se registran todos los recursos que se van a utilizar posteriormente en la aplicación, mediante una relación entre el propio recurso y un identificador que se asigna de forma automática para poder referirnos a ese recurso, de hecho cada vez que se hace una modificación dentro de los recursos por ejemplo si se añade cajas de texto, una imagen o se modifica algún *layout* esta clase se regenera automáticamente, teniendo siempre disponible nuestros recursos en nuestro código Java. La regeneración de esta clase puede ser un problema por el tiempo que cuesta cuando tenemos muchos recursos o una cantidad muy grande de *layouts* y de imágenes.

5.7.5. Almacén de datos

Android tiene un sistema de base de datos que es *sqlite3* que nos permite tener una pequeña base de datos dentro del teléfono, con toda la potencia de un gestor de base de datos permitiéndonos crear tablas, hacer inserciones, actualizaciones, consultas complejas con *joins*, etc. Por otro lado también tiene otro sistema de almacenamiento persistente que son las preferencias que nos sirven por ejemplo para almacenar de forma persistente el nombre de usuario para usarlo siempre de forma rápida en cualquier momento con la característica que este sigue ahí incluso si se sale y se vuelve a entrar en la aplicación.

5.7.6. Gestión de permisos y seguridad

Una aplicación para Android tiene un fichero de configuración bastante curioso llamado *Manifest,* en el que se listan los permisos y para cada elemento de desarrollo típico que puede ser problemático para la seguridad hay un permiso asociado que el usuario tiene que aceptar para poder instalar y usar la aplicación, por ejemplo un permiso para poder usar la cámara, otro para poder acceder a Internet, etc.

En el fichero *manifest* también hay que registrar todas las actividades que tenemos en la aplicación.

5.7.7. Pruebas de unidad

Para descubrir fallos en la aplicación existen diversas herramientas para el testeo que son test de integración o test unitarios, así por ejemplo nos permite coger una actividad y lanzar test de forma aleatoria sobre ella, entre las herramientas disponibles están Monkey y Robotium. Monkey de forma aleatoria y Robotium de forma guiada.

5.7.8. Reutilizar código escrito en C y en C++

El lenguaje C resulta más adecuado que Java para determinadas tareas, en especial aquellas que requieren procesos muy intensivos en cálculo, como pueden ser tratamiento de señales, procesamiento digital de imágenes o los algoritmos de cifrado.

Con el fin de incrementar la funcionalidad de las aplicaciones en Android, mediante el NDK es posible el desarrollo de aplicaciones en lenguaje C permitiéndonos escribir y crear aplicaciones nativas para Android y en muchos casos reutilizar código sin la necesidad de volverlo a escribir para esta plataforma.

El NDK compila código en lenguaje C y lo transforma en librerías nativas JNI exportando las funciones en C y dejándolas a disposición para utilizarlas con el código Java del SDK. Uno de los requisitos fundamentales para invocar a las funciones en C es que estas sigan un estricto esquema para nombrarlas que incluye los nombres de paquete, clase y método.

Capítulo 6. Diseño

El diseño de este tipo de aplicación nos recuerda a las tradicionales aplicaciones web donde la clave es determinar que lógica se ejecuta en el servidor y que lógica se ejecuta en el cliente, recordemos que en los primeros sitios web los clientes eran usados principalmente para ingresar y renderizar datos es decir todo el procesado de la información se realizaba en el lado del servidor, hoy en día gracias a tecnologías como *JavaScript*, *Ajax*, *Flash* y *Silverlight* los clientes realizan muchas tareas y procesos permitiendo así distribuir la carga de trabajo en ambos lados de la arquitectura cliente-servidor.

Hemos decidido implementar una aplicación móvil de reconocimiento facial en este sentido, así todo el proceso desde que el usuario mete una fotografía como dato de entrada y obtiene el resultado del reconocimiento como dato de salida, lo dividimos en sub procesos o módulos[2] que se puedan ejecutar tanto en el lado del servidor como en el lado del cliente.

Cuando usamos Internet requerimos de un navegador web, en este ingresamos una dirección de alguna pagina y después de unos pocos segundos nos muestra el contenido de la misma, en este proceso están inmersos cinco sub procesos ingreso de información, envió de la petición, procesamiento de la petición/creación de la respuesta, retorno de la respuesta y despliegue de información.

Los sub procesos de ingreso y despliegue de información obligatoriamente se realizan en el lado del cliente, el envió y retorno de datos son estándares especificados por el protocolo que las dos partes implicadas entienden y el sub proceso que más poder computacional requiere es el procesamiento en sí de la información en el lado del servidor que no representa ningún problema por sus potentes características, pero lo que si representa un problema es la comunicación entre el cliente y el servidor que en muchos de los casos llega a ser el cuello de botella en el flujo de la comunicación. Para aliviar un poco este problema optimizamos el uso de los recursos implicados en el envió de datos, refinando la petición es decir hacer un pre procesamiento de la información en el lado del cliente antes de enviarla hacia los servidores.

En este capítulo describimos la arquitectura, la interfaz, y la implementación de la aplicación "*FaceRec*" nombre con el cual la hemos bautizado.

6.1. Arquitectura

En la *figura 6.1* se muestran los sub procesos en los que esta dividida la aplicación: captura, pre procesado, envió, procesado, retorno y resultados con el flujo de datos entre los mismos.

Como hemos mencionado la captura y el despliegue de resultados se realizan obligatoriamente en el lado del cliente, el pre procesamiento es obligatorio para el procesado en modo local y una opción para el procesado en modo remoto en el cual se debe considerar que el envió y retorno de peticiones y respuestas entre cliente y servidor viajan a través de la red inalámbrica WiFi o 3G influyendo

2 Siguiendo la filosofía divide y vencerás.

directamente en el tiempo para la obtención del resultado.

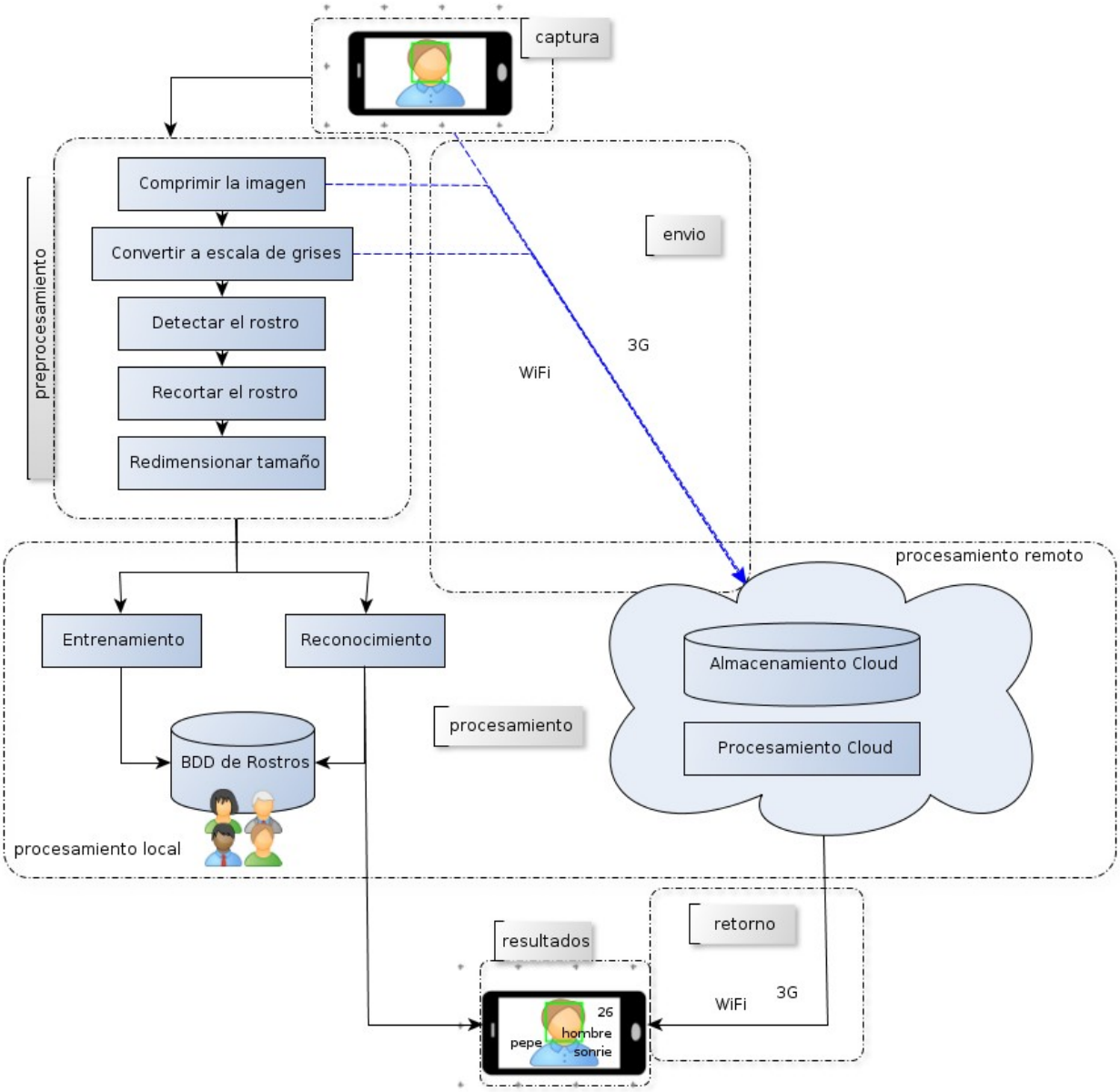


Figura 6.1. Arquitectura de la aplicación.

6.2. Interfaz de usuario

Como trabajamos con captura de imágenes utilizamos la disposición de pantalla en modo horizontal, en la parte derecha de la misma tenemos dos botones distintos uno para entrenar y otro para reconocer, las acciones que se ejecutan al presionar cualquiera de los dos botones activa un *timer* de cinco segundos para realizar fotografías de la persona con la cual vamos a interactuar en la aplicación, en el caso de entrenar se realizan cinco disparos y en el caso de reconocer se lanza uno solo. La *figura 6.2* muestra el esquema de la misma.

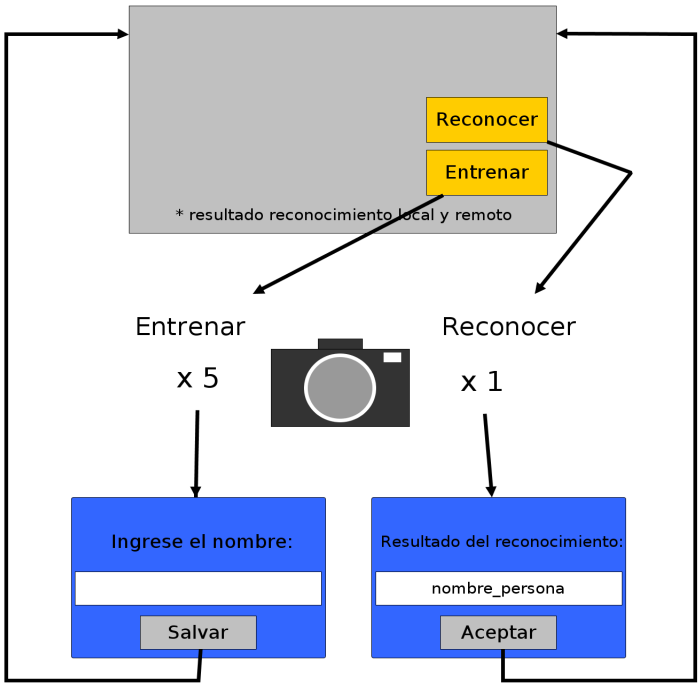


Figura 6.2. Interfaz de usuario

Después de haber capturado la imagen o las imágenes tenemos los datos de entrada para realizar su procesamiento según corresponda.

6.3. Sub procesos

El proceso de computo en general de ingresar un dato, procesarlo y obtener un resultado para nuestra aplicación lo hemos divido en cinco sub procesos: captura, pre procesado, envió, procesado, retorno y resultados. A continuación vamos a explicar que realiza cada sub proceso y como esta implementado dentro la aplicación.

6.3.1. Captura

Una vez que hemos iniciado la aplicación y al haber presionado sobre cualquiera de los dos botones iniciamos la captura de datos con los cuales vamos a trabajar, realizamos automáticamente cinco disparos con la cámara incorporada en el dispositivo móvil para utilizarlos en el entrenamiento y uno solo para el usarlo en el reconocimiento.

Los datos con los que vamos a trabajar son imágenes y pixels, las imágenes en color tienen tres canales uno para cada color RGB, las imágenes en escala de grises tienen un solo canal. Los pixels son la unidad mínima de información capturada y las dimensiones de una imagen se expresan en pixels de anchura por pixels de altura. Y tienen un valor mínimo de cero cuando es negro y un valor máximo de 255 cuando es blanco.

El SDK de Android ofrece varias formas de capturar una imagen, una de ellas es lanzar un *Intent* a la aplicación de la cámara para que esta haga su trabajo y la otra es implementarla mediante código para tener un mayor control de las configuraciones y de los parámetros, la segunda opción es la que utilizamos en esta aplicación y el proceso que debemos seguir lo podemos observar en la *figura 6.3*

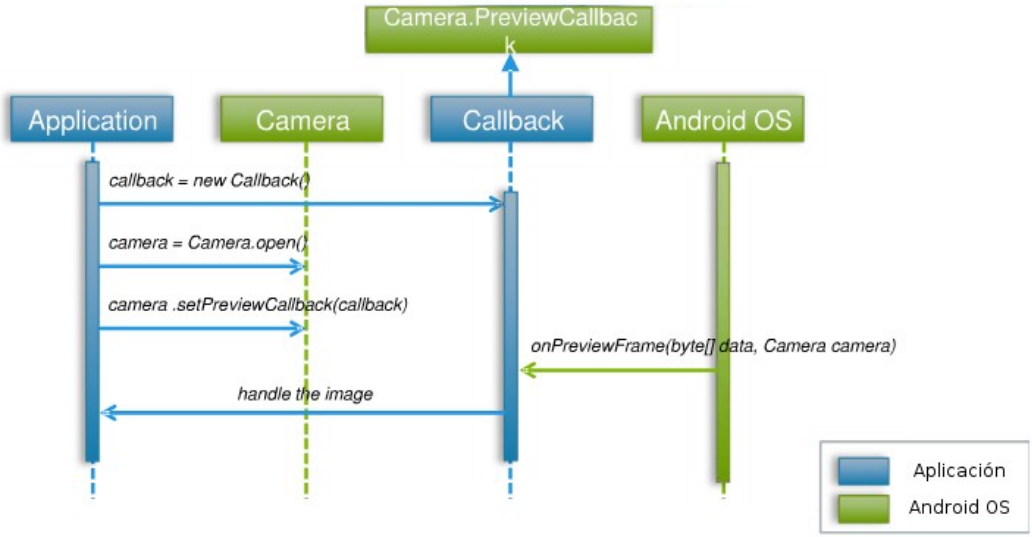


Figura 6.3. Adquisición de un frame de video de la cámara de un dispositivo Android

Para ello: en el *manifest* necesitamos dos permisos uno para realizar fotografías y otro para poderlas almacenar en la tarjeta SD.

```
<!-- permisos-->
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Lanza una llamada a la pre visualización de la cámara, cuando los datos de la foto están disponibles adquirimos la imagen, el código Java mas importante para realizar esto es:

```
TrainCallBack trainCallBack = new TrainCallBack();
mCamera = Camera.open();
mCamera.setPreviewDisplay(SurfaceHolder);
```

Para almacenar en la tarjeta SD convertimos la imagen que la tenemos en un *array* de bytes a un mapa de bits.

```
public void onPictureTaken(byte[] data, Camera camera) {
      Bitmap bitmap = BitmapFactory.decodeByteArray(data, 0, data.length);
      FileOutputStream outStream = null;
      outStream = new FileOutputStream(fileName);
      outStream.close();
}
```

6.3.2. Pre procesamiento

En esta etapa con la imagen capturada de la etapa anterior extraemos las principales características siguiendo un cierto patrón para facilitar su posterior procesamiento en modo local o reducir su tamaño para optimizar el envió al procesamiento en modo remoto.

El pre procesamiento está compuesto por: la compresión de la imagen, convertir la imagen en escala de grises, detectar rostros en una imagen, recortar el rostro detectado en la imagen y re-dimensionar la pequeña imagen del rostro a un tamaño estándar, revisemos cada una de ellas.

6.3.2.1. Comprimir la imagen

Comprimimos para utilizar menos memoria en las operaciones que efectuemos con la imagen y también para una rápida transmisión en Internet.

El SDK de Android nos permite crear una versión comprimida de la imagen a partir del flujo de datos capturado en la etapa anterior con la función *compress* en la que especificamos el formato que llevara la imagen comprimida, el porcentaje de compresión y el *stream* de datos a comprimir.

En nuestro caso utilizamos el formato JPG, con un nivel de compresión bajo para no perder los detalles y le pasamos el *stream* de datos obtenido en la captura.

```
Bitmap.compress(CompressFormat.JPEG, 80, outStream)
```

6.3.2.2. Convertir a escala de grises

Transformamos la imagen de color a una imagen en escala de grises, para trabajar con un solo canal descartando los canales de color RGB y convertir cada uno de los pixels para poder representarlos con unicamente ocho bits.

A partir de esta función comenzamos a utilizar la librería OpenCV integrada al SDK a través del NDK de Android, las cuales se escriben en lenguaje C++, en el anexo2 encontramos los principales tipos de datos definidos y utilizados en la aplicación así como tambien las principales funciones para trabajar con la librería OpenCV .

La función *cvCvtColor* convierte la imagen original de color a una imagen en escala de grises, primer y segundo parámetro respectivamente, si especificamos el código *CV_RGB2GRAY* en el tercer parámetro de la función.

```
cvCvtColor( src, dst, CV_RGB2GRAY )
```

6.3.2.3. Detectar el rostro

Identificar la ubicación de los rostros de las personas que aparecen en una imagen, esta funcionalidad la podemos encontrar implementada en el SDK de Android y también en OpenCV.

En el SDK de Android obtenemos esta funcionalidad mediante el objeto *FaceDetector*, para el cual son necesarios dos objetos de este tipo, el primero es un vector de tamaño igual al número máximo de rostros que le especifiquemos y el segundo indicándole las dimensiones de la imagen. Obtenemos un vector con listado de rostros identificados en la imagen *myBitmap* con la función *findfaces*.

```
vectorRostros=new FaceDetector.Face[ROSTROS_MAX];
detectorDeRostros=new FaceDetector(width,height,ROSTROS_MAX);
detectorDeRostros.findFaces(myBitmap,vectorRostros);
```

Una muestra de esta funcionalidad lo podemos ver en la *figura 6.4* en la cual donde además de los dos rostros hay una región en la parte derecha que también se considera como un rostro.

Figura 6.4. FaceDetector de Android

En la aplicación preferimos utilizar los clasificadores en cascada desarrollados para OpenCV, los clasificadores tienen esta estructura:

```
\begin{verbatim}
    Cascade:
        Stage,,1,,:
            Classifier,,11,,:
                Feature,,11,,
            Classifier,,12,,:
                Feature,,12,,
            ...
        Stage,,2,,:
            Classifier,,21,,:
                Feature,,21,,
            ...
        ...
\end{verbatim}
```

Y están almacenados en ficheros XML, estos clasificadores han sido entrenados previamente por los creadores de la librería y simplemente los utilizamos a nuestra conveniencia.

En concreto utilizamos el clasificador *haarcascade_frontalface_alt* para identificar rostros en vista frontal en toda la imagen, el clasificador devuelve un "1" si considera que hay algún parecido entre algún objeto y un rostro y un "0" en caso contrario, a continuación mostramos el inicio del clasificador en el cual consta el primer árbol con sus valores para aceptar o descartar rostros, el fichero contiene hasta 212 árboles unas 26 mil líneas similares a las que podemos ver a continuación:

```
<opencv_storage>
<haarcascade_frontalface_alt type_id="opencv-haar-classifier">
  <size>20 20</size>
  <stages>
    <_>
      <!-- stage 0 -->
      <trees>
        <_>
          <!-- tree 0 -->
          <_>
            <!-- root node -->
            <feature>
              <rects>
                <_>3 7 14 4 -1.</_>
                <_>3 9 14 2 2.</_></rects>
              <tilted>0</tilted></feature>
            <threshold>4.0141958743333817e-003</threshold>
            <left_val>0.0337941907346249</left_val>
            <right_val>0.8378106951713562</right_val></_></_>
        <_>
          <!-- tree 1 -->
          <_>
            <!-- root node →
...
```

No necesitamos tocar este fichero simplemente lo utilizamos para ello lo incluimos en los recursos de la aplicación junto a los *layouts* y demás y lo cargamos cuando ejecutemos la aplicación:

```
BufferedInputStream haarXmlReader = new BufferedInputStream(
     this.getResources().openRawResource(R.raw.haarclassifier));
```

y para utilizarlo en C:

```
Cascada = (CvHaarClassifierCascade *) cvLoad(haarClassifierXml.c_str(), 0, 0,
0);
```

El clasificador lo utilizamos mediante la función *cvHaarDetectObjects* que se encarga de encontrar regiones rectangulares en una imagen dada que más se asemeje a los objetos contenidos en el clasificador, retornando las regiones que pasaron la clasificación en cascada como una secuencia de rectángulos, la función escanea la imagen varias veces a diferentes escalas solapando regiones y volviendo a aplicar los clasificadores sobre estas regiones, para reducir el número de regiones analizadas se aplica algunas heurísticas, los parámetros por defecto son factor_escala=1.1, mínimo_vecinos=3

```
CvSeq *pSeqRectFace = cvHaarDetectObjects(
            InputImg,   //imagen a escanear
            Cascada,    //fichero del clasificador en cascada
            cvCreateMemStorage(0),//espacio en memoria para almacenar el
resultado
            1.1,        //factor de escala para los siguientes escaneos
            3,          //número mínimo de rectángulos vecinos
            CV_HAAR_FIND_BIGGEST_OBJECT   //heurística
            cvSize(0, 0));  //tamaño
```

Esta etapa finaliza con el puntero *pSeqRectFace* que está apuntando a la secuencia de rectángulos de los rostros identificados en la imagen, este puntero es con el cual vamos a trabajar en las etapas siguientes.

6.3.2.4. Recortar el rostro en la imagen

Para reducir el área de trabajo recortamos la imagen eliminando las regiones innecesarias para utilizar únicamente el rostro.

Para obtener unicamente el rostro en una imagen llamamos a la función *cropImage* pasándole como parámetros la imagen en escala de grises y la secuencia de rectángulos de rostros obtenidos anteriormente:

```
IplImage* croppedImage = cropImage(greyImage, RectFace);
```

En la función *cropimage* creamos dos imágenes una temporal para crear una copia de la imagen original donde se configura la región de interés que es el rectángulo en el cual se encuentra el rostro y la copiamos a una segunda imagen de tamaño igual a la región de interés, la función retorna esta imagen que es la versión recortada de la imagen original.

```
IplImage* cropImage(IplImage *img, CvRect region) {

      IplImage *imageTmp;
      IplImage *imageRecortada;
      CvSize size;

      // Primero creamos una nueva imagen temporal y copiamos el contenido de img
      size.height = img->height;
      size.width = img->width;
      imageTmp = cvCreateImage(size, IPL_DEPTH_8U, 1); //8 bits x pixel y un canal
      cvCopy(img, imageTmp, NULL); //imagen origen a imagen destino sin cambios

      //establecemos la region de interes con el rectangulo
      //que contiene la cara sobre la imagen temporal
      cvSetImageROI(imageTmp, region);    //ROI region of interest

      //Creamos la nueva imagen que va a contener unicamente el rostro
      size.width = region.width;
      size.height = region.height;
      imageRecortada = cvCreateImage(size, IPL_DEPTH_8U, 1);
      cvCopy(imageTmp, imageRecortada, NULL);

      cvReleaseImage(&imageTmp);          //liberamos la imagen temporal
      return imageRecortada;
}
```

6.3.2.5. Re dimensionar el tamaño de la imagen recortada

Con el objetivo de asegurar que las imágenes de los rostros obtenidas en etapa anterior mantengan dimensiones homogéneas re dimensionamos su tamaño, así si la imagen es muy grande reducimos sus medidas y si es muy pequeña la ampliamos, basados en transformaciones geométricas con interpolaciones.

OpenCV trae la función *cvResize* a la cual le pasamos como parámetros la imagen de origen, la imagen con el tamaño deseado y el método de interpolación:

```
void cvResize (CvArr* src, CvArr* dst, inter )
```

Los métodos de interpolación para calcular el valor de un pixel depende de otro o de otros pixels como lo podemos observar en la *tabla 3*:

Interpolación:	En OpenCV	Método / resultados
Vecino más próximo	CV_INTER_NN	El color de cada pixel toma del vecino más cercano, es un método rápido pero se consigue una imagen tosca con efecto diente de sierra.
Bilineal	CV_INTER_LINEAR	El color de cada pixel se calcula como el color promedio de los cuatro pixels vecinos más cercanos. Un buen compromiso entre calidad y velocidad.
Bicúbic	CV_INTER_CUBIC	El color de cada pixel se calcula como el color promedio de los ocho pixels vecinos mas cercanos, necesita más tiempo a cambio de una mejor calidad.
Supermuestreo	CV_INTER_AREA	El color de cada pixel resulta de aplicar varias veces la transformación y tomar la media, logra un resultado de mucha más calidad y es computacionalmente mas costoso.

Tabla 3. Métodos de interpolación

```
IplImage* sizedImg = resizeImage(croppedImage, resizeWidth, resizeHeight);
```

En la función *resizeImage* le pasamos la imagen recortada anteriormente junto con la anchura y altura que se desea que tenga la nueva imagen para ello dependiendo del tamaño de la imagen original si es menor al tamaño deseado la ampliamos utilizando el método de interpolación bilineal y en el caso de que la imagen original es mayor al tamaño que el deseado la reducimos con la interpolación de súper muestreo.

```
IplImage* resizeImage(IplImage *origImg, int newWidth, int newHeight) {
      IplImage *outImg;
      CvSize sizeOriginal;
      CvSize sizeNew;

      sizeOrig.height = origImg->height;
      sizeOrig.width = origImg->width;

      sizeNew.height = newHeight;
      sizeNew.width = newWidth;

      cvResetImageROI((IplImage*) origImg);
      outImg = cvCreateImage(sizeNew, IPL_DEPTH_8U, 1);

      if (sizeNew > sizeOriginal) {
            // Ampliar la imagen
            cvResize(origImg, outImg, CV_INTER_LINEAR);
```

```
	} else {
		// Reducir la imagen
		cvResize(origImg, outImg, CV_INTER_AREA);
	}
	return outImg;
}
```

6.3.2.6. Ecualizar

Para situaciones en que nos encontremos con imágenes oscuras o con bajo contraste, la ecualización nos permite realzar el contraste y resaltar detalles difíciles de ver, un ejemplo de ecualización lo podemos ver en la *figura 6.5*

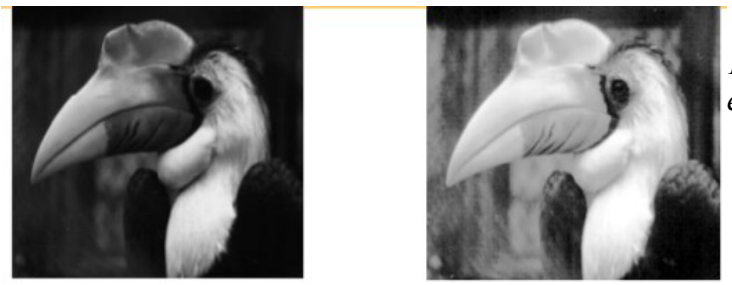


Figura 6.5. Efecto de ecualizar una imagen oscura

A la función *cvEqualizeHist* simplemente le pasamos la imagen obtenida en la etapa anterior y la imagen donde se almacenara el resultado.

```
IplImage* equalizedImg = cvCreateImage(cvGetSize(sizedImg), 8, 1);
cvEqualizeHist(sizedImg, equalizedImg);
```

6.3.3. Procesamiento

El procesamiento se realiza en dos modos distintos en local y en remoto. Utilizamos como entrada los datos pre procesados en los procesos de entrenamiento y reconocimiento. En modo local mediante la aplicación realizamos ambos procesos y en modo remoto nos conectamos mediante la API a los servicios de reconocimiento de *face.com* para obtener varios tipos de reconocimiento facial como identidad, genero, edad y estado de ánimo, en este modo nosotros no hacemos entrenamientos ellos ya lo tienen hecho, revisemos a detalle cada uno de ellos.

6.3.3.1. Procesamiento en modo local

Utilizamos los datos que han sido pre procesados para realizar tanto el entrenamiento como el reconocimiento.

a.) Entrenamiento

El entrenamiento consta de dos partes muy importantes: el análisis de componentes principales y la descomposición eigen, como lo podemos ver en la *figura 6.6*

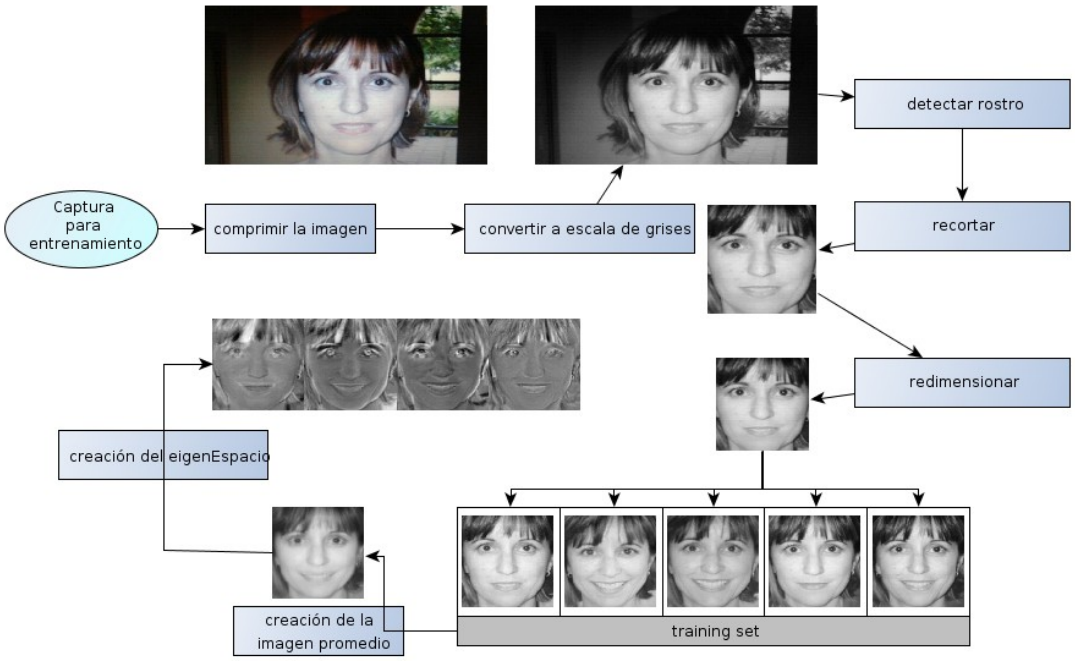


Figura 6.6. Etapas del pre procesado y proceso de entrenamiento

En el PCA el objetivo es reducir la dimensionalidad de los datos perdiendo la menor cantidad posible de información, en OpenCV lo podemos realizar mediante la llamada a la función *cvCalcEigenObjects* que calcula de forma iterativa las correlaciones entre las variables de un grupo de objetos devolviéndonos la imagen promedio que básicamente es una imagen borrosa de un rostro pero que representa a cualquier imagen en el *dataset* junto con los valores y vectores únicos (*eigenvalues and eigenvectors*).

Para ello en el código: *code*_a.) cargamos las imágenes, *code*_b.) preparamos las estructuras de datos para almacenar los resultados y *code*_c.) aplicamos la función *CalcEigenObjetcs*:

*code*_a.)

```
int numImages = trainingImages.size(); // numero total de imágenes entrenadas;
numEigens = numOfImgs - 1;

for (int i = 0; i < numImages; i++) {
     faceImageArray[i] = cvLoadImage(imgFileName, CV_LOAD_IMAGE_GRAYSCALE);
}
```

*code*_b.)

```
CvSize faceImageSize;
faceImageSize.width = this->faceImageArray[0]->width;
faceImageSize.height = this->faceImageArray[0]->height;

for (i = 0; i < numEigens; i++) {
     eigenVector[i] = cvCreateImage(faceImageSize, IPL_DEPTH_8U, 1);
}

AvgTrainImg = cvCreateImage(faceImageSize, IPL_DEPTH_8U, 1);
eigenValoresMat = cvCreateMat(1, nEigens, CV_32FC1);
```

*code*_c.)

```
cvCalcEigenObjects(numImages,           //numero de objetos a calcular
                   (void*) faceImageArray,      //puntero al arreglo de imágenes
IPL
                   (void*) eigenVector,    //puntero al arreglo de eigenObjetos
                   CV_EIGOBJ_NO_CALLBACK,  //bandera de I/O
                   0,0,  //tamaño del buffer en bytes de I/O, cero es desconocido
                   calcLimit,  //criterio para detener los cálculos de
eigenObjetos
                   AvgTrainImg,            //objeto promedio
                   eigenValoresMat->data.fl); //puntero a la matriz de
eigenValores

cvSaveImage(avgImgPath, AvgTrainImg);    //guardamos la imagen promedio
```

Una vez que tenemos los componentes principales los representamos en una matriz y cada uno de los elementos en esta matriz representa los coeficientes factoriales de las variables, es decir la correlación que existe entre las variables y las componentes principales. En OpenCV se realiza mediante la función *cvEigenDecomposite* que recibe como parámetros el número de objetos eigen, el vector eigen, la imagen promedio con la cual se van a calcular los eigenfaces y la matriz que va a contener los coeficientes en tantas columnas como número de objetos eigen y tantas filas como imágenes entrenadas.

Para ello en el código realizamos: *code_d*.) creamos la matriz, *code*_e.) la descomposición eigen y *code*_f.) creamos el espacio eigen.

code_d.)

```
MatrixEntrenos = cvCreateMat( trainingImages.size(),       //filas
                              numEigens,             //columnas
                              CV_32FC1);  //tipo de elemento de la matriz
                                          //32 bit float con signo con un canal
```

code e.)

```
for (i = 0; i < trainingImages.size(); i++) {

      cvEigenDecomposite(faceImageArr[i],
                         nEigens,          //numero de objetos eigen
                         eigenVector,      //puntero al vector eigen
                         0, 0,             //ioFlag
                         AvgTrainImg,      //la imagen promedio
                         MatrixEntrenos->data.fl + i); //calculo de coeficientes
                  }
```

code_f.)

```
// Crear una gran imagen con muchas imágenes eigenface
// Poner todas las eigenfaces una junto a otra y hasta 8 en una fila
      int nCols = min(numEigens, 8);
      int nRows = 1 + (numEigens / 8); // Poner el resto en nuevas filas.
      int w = eigenVector[0]->width;
      int h = eigenVector[0]->height;

      CvSize size;
      size = cvSize(nCols * w, nRows * h);
      IplImage *granImagen = cvCreateImage(size, IPL_DEPTH_8U, 1);

      for (int i = 0; i < numEigens; i++) {
            IplImage *ImgEig = (eigenVector[i]);
                                    // Pegarla en la posición correcta
            int x = w * (i % 8);
            int y = h * (i / 8);
            CvRect ROI = cvRect(x, y, w, h);
            cvSetImageROI(granImagen, ROI);
            cvCopyImage(ImgEig, granImagen);
            cvResetImageROI(granImagen);
      }
      cvSaveImage(Path, granImagen);
```

en el anexo 3 se muestran las imágenes que se obtienen al juntar todas las imágenes eigenfaces en una sola, las primeras tienen las característica dominantes y las ultimas son imágenes con características débiles, prácticamente imágenes con mucho ruido.

b.) Reconocimiento

Para reconocer una persona en una nueva imagen que ha pasado por las etapas de pre procesamiento y considerando que ya se han realizado entrenamientos previamente, esta imagen es llamada imagen de test de la cual vamos a extraer los coeficientes para ello se aplica nuevamente el análisis de componentes principales de OpenCV con la misma función utilizada en el entrenamiento *cvEigenDecomposite* e incluso los mismos parámetros: número de objetos eigen, el vector eigen, la imagen promedio excepto que en esta no le damos la matriz para que guarde los resultados sino simplemente el puntero al objeto que va contener los coeficientes.

Una vez obtenidos los coeficientes en un listado se compara con los coeficientes representativos de todas las imágenes de entrenamiento almacenados en la matriz, la comparación consiste en hacer la diferencia entre estos coeficientes y elegir siempre el valor mínimo.

Con el identificador del coeficiente con el que obtenemos el valor mínimo sabemos a qué imagen le pertenece dentro del vector de entrenamiento y podemos tomar el nombre de la persona. Probablemente no sea el correcto debido a semejanzas entre rostros, problemas con diferentes iluminaciones o ruido en las imágenes para lo cual se ofrece además un valor de indica el nivel de confianza del reconocimiento que está basado en el cálculo de la distancia Euclidea con valores de entre menos uno y uno, que han sido escalados en el rango de cero y uno. Así imágenes similares tienen un nivel de confianza de 0.5 a 1 y las imágenes diferentes tienen una confianza entre 0 y 0.5.

Para esto en el código realizamos: *code*_g.) descomposición eigen y *code_h*.) comparación de coeficientes de la imagen con los de la matriz y el cálculo del nivel de confianza.

code_g.)

```
float * coefTestFace;

cvEigenDecomposite(equalizedImg,
                   numEigens,        //numero de objetos eigen
                   eigenVector,      //puntero al vector eigen con los obj input
                   0, 0,             //ioFlag
                   AvgTrainImg,      //la imagen promedio
                   coefTestFace);    //calculo de coeficientes
```

code_h.)

```
int ubicacion = 0;
double distCuadMinina = 1e12;


for (int imgEntreno = 0; imgEntreno <  trainingImages.size(); imgEntreno++) {
      double distanciaCuadrado= 0;

      for (int i = 0; i < numEigens; i++) {
            float d_i = coefTestFace[i] - MatrixEntrenos
                                    ->data.fl[imgEntreno * numEigens + i];

            distanciaCuadrado+= d_i * d_i; // distancia Euclidea.

            if (distanciaCuadrado <  distCuadMinina) {
                  distCuadMinina = CuadradoDistancia;
                  ubicacion = imgEntreno;
            }
      }

      // el nivel de confianza basado en la distancia Euclidea,
      float confianza = 1.0f - sqrt(distCuadMinina /
                     (float) (trainingImages * numEigens)) / 255.0f;

      string personName = trainingImages[ubicacion].first;
}
```

6.3.3.2. Procesamiento en modo remoto

El procesamiento en modo remoto lo delegamos a *face.com* mediante la utilización de su API. El API proporciona una abstracción del sistema y pone los servicios que ofrece la empresa a disposición de los desarrolladores que los vamos a utilizar por medio de la aplicación

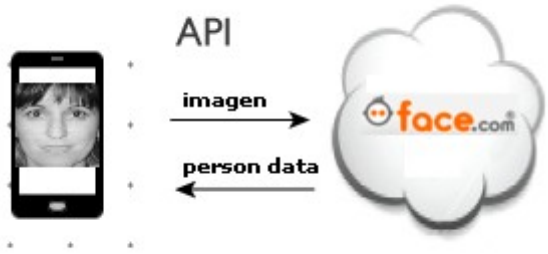


Figura 6.7. Esquema de uso del API de face.com

Primeramente solicitamos una clave de acceso a la API en *developers.face.com* para ello nos registramos gratuitamente, aceptamos los términos del uso, registramos el nombre de la aplicación y conseguimos las credenciales de acceso, que son el par: <clave, secreto> llamadas API_Key y API_Secret ambos son cadenas de 33 caracteres. Así, todas las consultas mediante API incorporan la clave y el secreto como elemento de seguridad y para controlar el uso moderado de los servicios.

Entre el amplio abanico de servicios que actualmente ofrece *face.com* los cuales podemos acceder mediante la API son: almacenamiento temporal de imágenes para el reconocimiento facial, reconocimiento de género, reconocimiento de estado de ánimo, estimación de edad y continuamente desarrollan más funcionalidades.

A estos servicios accedemos de la misma manera que lo hacemos con los servicios web de toda la vida, en general creamos el cliente, creamos la petición, efectuamos la petición y procesamos la respuesta.

Así todas las consultas al API se realizan siguiendo el siguiente formato:

```
http://URL_BASE/{metodo}.{formato}/{parametros}
```

donde:

la `URL_BASE` es:
"http://api.face.com"
el `metodo.formato` pueden ser:
RECOGNIZE("/faces/recognize.json")
DETECT("/faces/detect.json")
TRAIN("/faces/train.json")

los parámetros obligatorios en las peticiones son:
la clave ("api_key", apiKey);
el secreto ("api_secret", apiSecret);
los opcionales :
la url ("urls", urls);
credenciales redes sociales ("user_auth", creds.getAuthString());

En la aplicación utilizamos el paquete *org.apache.http* para la gestión de comunicaciones HTTP y las clases *entity*, *response*, *client*, *mime* para gestionar las conexiones, respuestas, clientes, tipos de datos, etc y mediante los métodos *get* y *post* para leer y escribir recursos respectivamente.

En el *manifest* necesitamos dos permisos más uno para acceder a Internet y verificar el tipo de red inalámbrica al que nos conectamos.

```
<!-- mas permisos-->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

La petición es de tipo POST que lleva la imagen pre procesada junto los parametros obligatorios para ello es necesario una entidad *MultipartEntity* a la cual mediante el método *addPart* le añadimos el fichero de la imagen y los parámetros.

Efectuamos la petición POST mediante *httpClient.execute,* la imagen es almacenada en los servidores de *face.com* temporalmente con acceso público y sobre esta URL se ejecutan los algoritmos de reconocimiento y estimación. Dependiendo de la red a la que estamos conectados al momento de enviar la petición esta demorara más o menos tiempo en obtener la respuesta.

El código lo hacemos mediante:

```
final MultipartEntity entity = new MultipartEntity();

entity.addPart("image", new FileBody(file));

for (NameValuePair nvp : params) {
      entity.addPart(nvp.getName(), new StringBody(nvp.getValue()));
}

postMethod.setURI(uri);
postMethod.setEntity(entity);

final HttpResponse response = httpClient.execute(postMethod);
```

La respuesta que retorna es una cadena en formato JSON, el tratamiento de esta es parte del sub proceso siguiente.

El detalle que como se realiza el reconocimiento y que algoritmos utilizan en *face.com* no son datos de carácter público ni es código abierto, es como una caja negra a la que no podemos abrir, sin embargo en [11] nos da nociones que utilizan reconstrucciones en 3D de los rostros para realizar todos los reconocimientos y estimaciones. Nosotros simplemente somos clientes que consumimos el

API que nos abstrae de toda la complejidad de los procesos que se realizan por debajo.

a.) Reconocimiento y Entrenamiento

Podemos utilizar el método: *faces.detect* para obtener el reconocimiento de género, expresión facial y estimación de la edad simplemente enviando la imagen y nuestras credenciales.

Y para el entrenamiento y reconocimiento están ligadas a las redes sociales en las que se puede enviar las credenciales de usuario tanto de Facebook como de Twitter y los entrenamientos se realizan con las imágenes publicadas y las etiquetas de los amigos y la información se comparte con todos los usuarios de la misma API_KEY. Se puede acceder a estas funcionalidades mediante *faces.recognize* y *faces.train*, esta funcionalidad no esta implementada en la aplicación actualmente tal vez si para futuras versiones de la aplicación.

6.3.4. Envió y retorno

En modo local son solo llamadas a funciones internas y el tiempo en realizarlas es despreciable. Al contrario de lo que sucede en el procesamiento en modo remoto que son factores muy a tomar en cuenta los paquetes viajan a través del protocolo HTTP que depende totalmente de la infraestructura de la red inalámbrica utilizada bien sea WiFi o 3G y el rendimiento de ambas redes puede verse fácilmente afectado por el trafico y otros factores.

6.3.5. Tratamiento de los resultados

Terminamos el proceso mostrando en pantalla los resultados obtenidos:

En modo local dibujamos un rectángulo de color verde sobre cada uno de los rostros identificados y almacenados en la secuencia *pSeqRectFace* y además se muestra el nombre de la persona identificada.

En el modo remoto realizamos el tratamiento haciendo un *parsing* de la respuesta tipo JSON a tipos de datos sencillos como *strings* o enteros, una respuesta JSON que se obtiene al invocar a los servicios de *face.com* la mostramos a continuación:

```
{
"mouth_right":{"y":35.17,"x":24.72},
"mouth_left":{"y":35.14,"x":21.32},
"recognizable":true,"chin":null,
"mouth_center"{"y":35.06,"x":23.05},
"width":10.45,
"label":"",
"ear_left":null,
"tagger_id":null,
"eye_left":{"y":28.95,"x":20.74},
"threshold":null,
"center":{"y":32.1,"x":22.9},
"manual":false,
"yaw":-8.13,
"roll":0.59,
"uids":[],
"height":13.93,
"confirmed":false,
"gid":null,
"ear_right":null,
"attributes":{
      "glasses":{"value":"true","confidence":94},
      "gender":{"value":"male","confidence":74},
      "face":{"value":"true","confidence":97},
      "smiling"{"value":"false","confidence":98},
      "age_est" {"value": 19,"confidence": 92}
},
"tid":"TEMP_F@1b58ae8490_32.10_0_1",
"pitch":4.81,
"nose":{"y":32.06,"x":23.25},
"eye_right":{"y":28.88,"x":25.16}
}
```


La respuesta contiene las coordenadas de los puntos en la imagen en que se ha encontrado cada parte del rostro muy útil si se desea darle un trato especial a los mismos, nosotros simplemente nos centramos en los atributos donde se encuentran los datos que nos interesan, para ello hacemos un *get* del atributo deseado:

```
gender = JSONObject.getString("value"));
```

Capítulo 7. Pruebas

Las pruebas consisten en la ejecución de la aplicación con el objetivo de obtener los tiempos de ejecución medios de cada uno de sus sub procesos en distintos modos: en el primero todos son ejecutados en modo local, posteriormente utilizamos el procesamiento en modo remoto y por último la combinación entre pre procesado en local y procesado en remoto, como lo podemos ver en el diagrama de flujo de la *figura 7.1.*

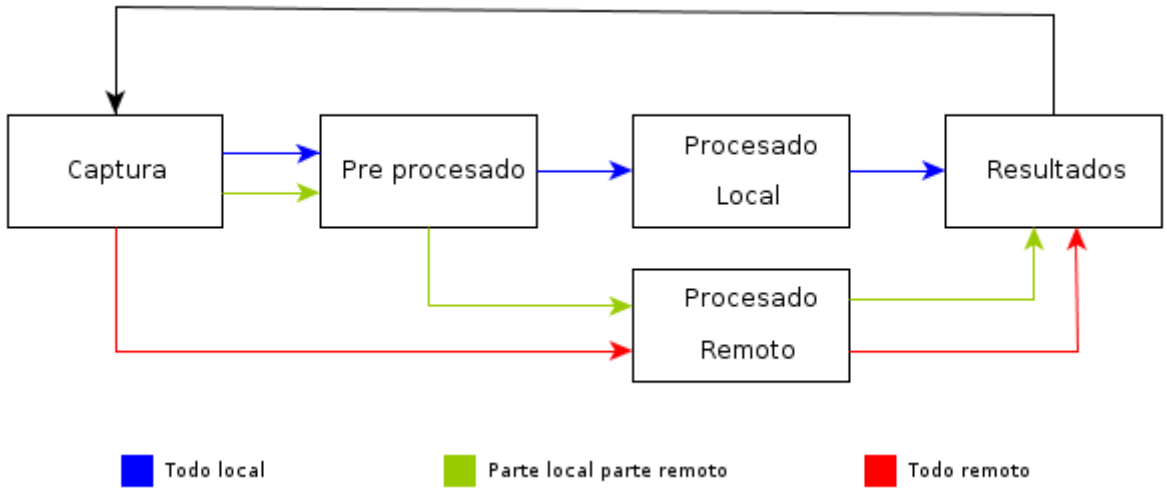


Figura 7.1. Diagrama de flujo de los modos de ejecución realizados en las pruebas

La aplicación Android ha sido desarrollada e instalada en un teléfono móvil *Samsung Galaxy Ace* [28], conectado a Internet a través de la red inalámbrica WiFi "UPVnet2G" del campus y también usando la conexión 3G de empresas comerciales de telefonía móvil, estas pruebas se realizaron por las noches evitando un poco la congestión que presentan estas redes en horas pico.

Para el *testing* de la aplicación tanto en modo local como en remoto necesitamos un conjunto de imágenes, en la primera parte describimos donde las conseguimos y sus características principales. luego presentamos los detalles de estas ejecuciones, posteriormente realizamos pruebas modificando el comportamiento del canal con distintos valores de anchos de banda, latencias y tasas de errores, seguido por un análisis del consumo de energía y finalizamos el capitulo con los resultados obtenidos.

Después de ejecutar todo este conjunto pruebas, mediante el análisis de los resultados tenemos que ser capaces de contestar a la pregunta de ¿para que valores es razonable hacer computo local y para que valores es conveniente hacer computo remoto?

7.1. Adquisición de imágenes

Existen una gran cantidad de bases de datos con imágenes de rostros que se utilizan para probar el funcionamiento de los nuevos algoritmos de reconocimiento que surgen en cada momento. Estas bases de datos son públicas y se pueden encontrar disponibles en la red en [23], Para nuestras pruebas hemos utilizado la base de datos de *Caltech* [24] que contiene una colección de fotografías de 896x592 pixels en formato JPEG de 27 personas en vista frontal con diferentes expresiones faciales, fondos y luminosidad.

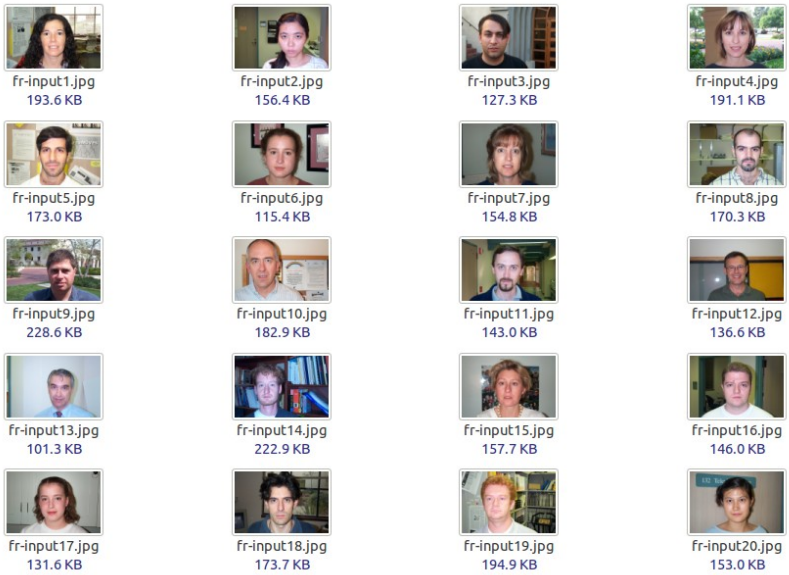


Figura 7.2. Distintas personas que componen la base de datos de Caltech

En *figura 7.2 y figura 7.3* se muestran unas imágenes a modo de ejemplo pertenecientes a la base de datos de rostros Caltech de las 27 personas utilizamos solo 20 de ellas por la gran cantidad de imágenes distintas de una misma persona.

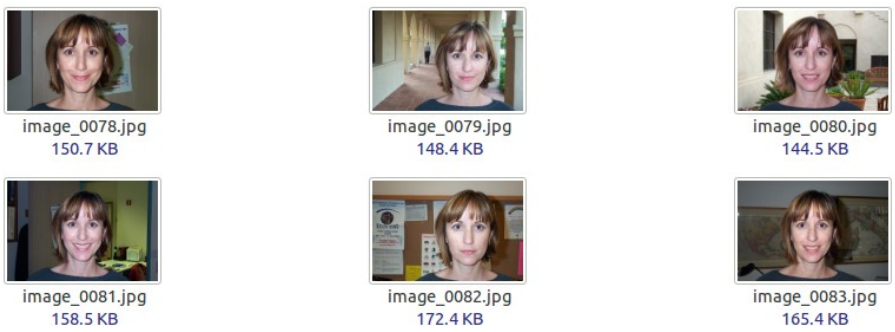


Figura 7.3. Distintas imágenes de una misma persona de la base de datos de Caltech

7.2. Procesamiento en modo local

7.2.1. Entrenamiento

Antes de reconocer un rostro, el sistema tiene que ser entrenado, en nuestro caso el entrenamiento es el proceso de capturar cinco fotografías del rostro de una persona para formar el conjunto de entrenamiento (*training set*) se precisa que las capturas sean de rostros en vista frontal, la interfaz de usuario se encarga de disparar automáticamente las capturas con un *timer* de cinco segundos entre cada una, después de la quinta captura escribimos un nombre para identificar a la persona que acabamos de entrenar con la aplicación, este proceso lleva alrededor de treinta segundos, pero cabe destacar que el *timer* es configurable a cualquier valor, se ha escogido cinco segundos para dar tiempo a conseguir mejores capturas.

Internamente cada persona que entrenemos en la aplicación tiene un vector *eigenfaces* el cual contiene los coeficientes representativos de está persona, si entrenamos la misma persona dos veces tenemos dos vectores diferentes para está misma persona pero los valores de los coeficientes deben ser próximos o muy similares.

El vector que contiene todas las imágenes de todas las personas que hayamos entrenado en la aplicación, en el primer uso de la misma el tamaño es cero e ira creciendo conforme vamos añadiendo personas a la aplicación, hemos definido que una persona puede tener hasta cinco imágenes en este vector, en los *tests* hemos llegado a almacenar hasta 20 personas con 5 imágenes de cada una ellas, obteniendo un vector con un tamaño igual a 100 imágenes.

Los entrenamientos más destacables fueron realizados con: 5, 10 y 20 personas, variando el número de imágenes para cada una de ellas desde 1 hasta 5.

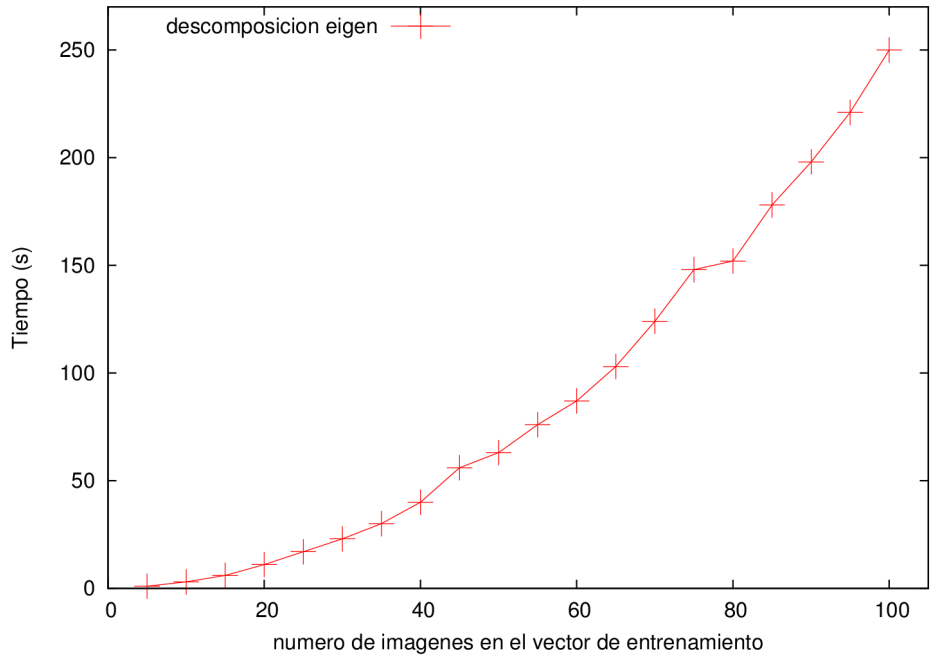


Figura 7.4.
Tiempos necesarios para realizar la descomposición eigen según el número de imágenes en el vector de entrenamiento

El tiempo para la descomposición eigen de las imágenes de todas las personas aumenta proporcionalmente con el tamaño del vector, así por ejemplo en el entrenamiento de 20 personas con una imagen de cada una de ellas obtenemos un vector con 20 imágenes y la descomposición eigen para este vector dura 10 segundos aproximadamente. En otro entrenamiento de 20 personas con 5 imágenes de cada una de ellas obtenemos un vector de 100 imágenes y la descomposición eigen para este vector requiere un tiempo alrededor de 4 minutos. Los tiempos medios de las pruebas realizadas se muestran en la *tabla 5* y gráficamente en la *figura 7.4.*

El conjunto de las imágenes del vector puede estar constituido por las combinaciones que consideremos adecuadas por ejemplo: 10 personas con 3 imágenes de cada una de ellas, para una mayor confianza en los resultados de reconocimiento se sugiere que cada persona posea las cinco imágenes.

7.2.2. Reconocimiento

El proceso de reconocimiento consume mucho menos tiempo comparado al proceso de entrenamiento, para cualquier persona cuya imagen ha sido entrenada de antemano, el reconocimiento facial se realiza en aproximadamente de 1,5 a 2,5 segundos.

El hecho de usar eigenfaces para el reconocimiento lo convierte en un proceso muy rápido y además permite la funcionalidad de operar en lotes de rostros en tiempos muy cortos [6].

Específicamente en nuestro caso el tiempo que necesita la aplicación para el reconocimiento de un rostro entre 5 personas es de 1,8 segundos y si aumentamos el número de personas a 20 el reconocimiento tarda alrededor de 2,5 segundos, como vemos en la *figura 7.5.*

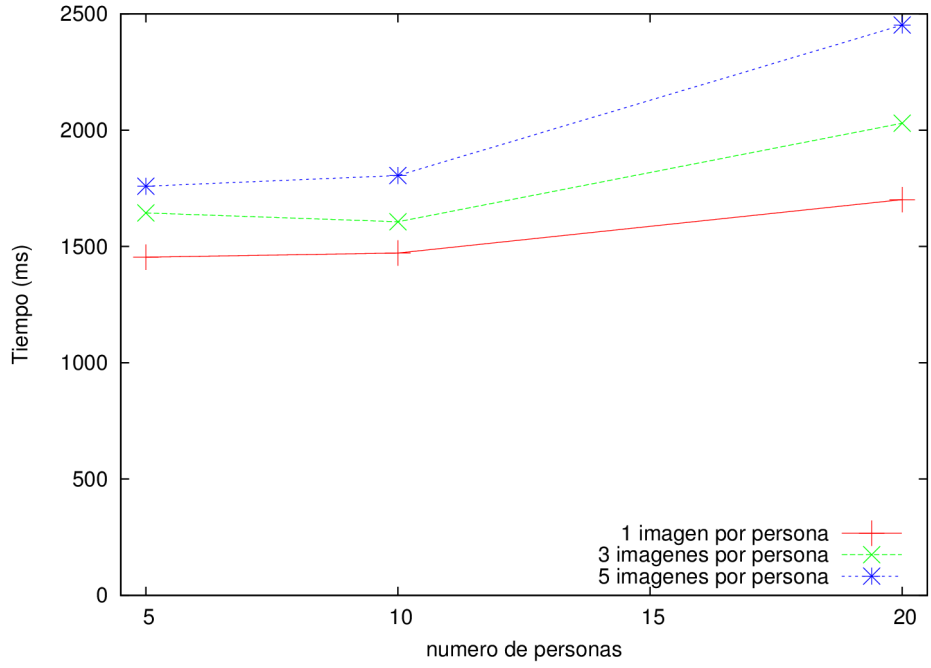


Figura 7.5. tiempo que toma realizar el reconocimiento de un rostro según el numero de personas y el numero de imágenes para cada una de ellas.

En el resultado que obtenemos además del nombre de la persona tenemos un nivel de confianza que está basado en la distancia Euclidea [21], así las imágenes similares tienen una confianza entre el 50 y 100 por cien y las imágenes distintas o diferentes una de confianza entre 0 y el 50 por cien.

Como podemos observar en la *figura 7.6*, con la línea roja tratamos de realizar reconocimientos con una sola imagen de entrenamiento por persona y como es de esperar obtenemos reconocimientos erróneos con valores de confianza inferiores al cincuenta por cien.

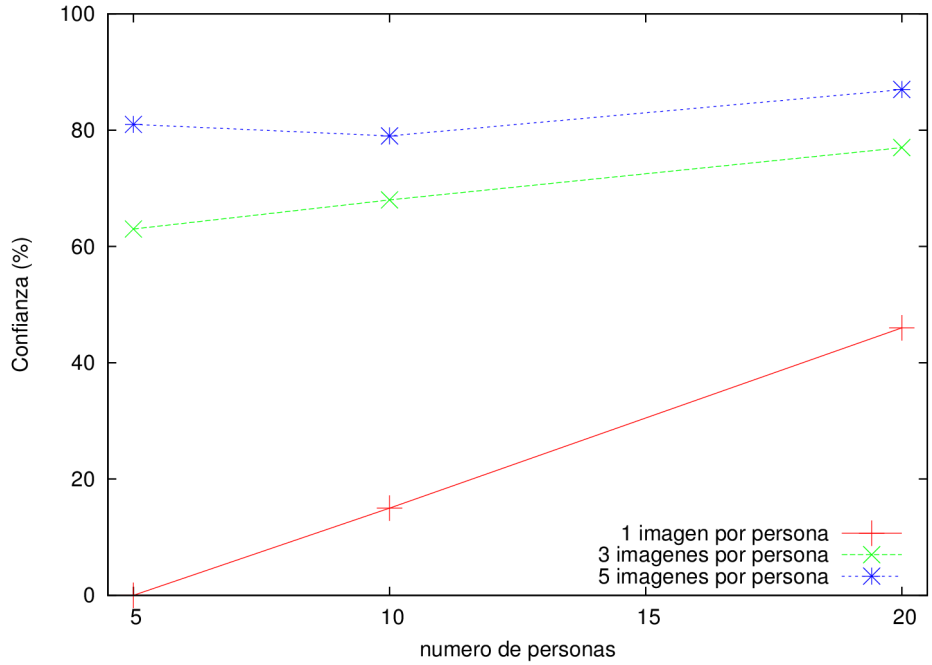


Figura 7.6. Confianza en el reconocimiento de un rostro según el numero de personas y el numero de imágenes para cada una de ellas

7.3. Procesamiento en modo remoto

Consiste en enviar la carga de trabajo a la empresa *face.com* especializada en el reconocimiento facial que ofrecen el reconocimiento como servicio para analizar las imágenes básicamente realizando las mismas funciones de reconocimiento que hacemos en el móvil pero con la gran diferencia que ellos ejecutan sus aplicaciones en potentes servidores, lo que les permite ofrecer un servicio totalmente escalable y confiable.

Face.com además de las funciones de reconocimiento facial ofrece otras como reconocimiento de género, reconocimiento de estado de ánimo y estimación de la edad, todo esta información es obtenida únicamente a partir de los rostros de las imágenes independientemente de la posición en la que se encuentren.

El envió de imágenes para su procesamiento en *face.com* se ha realizado variando el nivel de compresión en la aplicación para obtener ficheros de imagen con tamaños de: 7, 15, 30, 45, 70, 150 kilo Bytes (kB) y enviados a través de las redes WiFi y 3G.

Como se muestra la *tabla 7* y gráficamente en la *figura 7.7* con WiFi obtenemos unos resultados muy buenos independientemente del tamaño de la imagen de 1 a 2 segundos mientras que mediante la conexión móvil 3G tenemos tiempos desde 4 hasta 10 segundos.

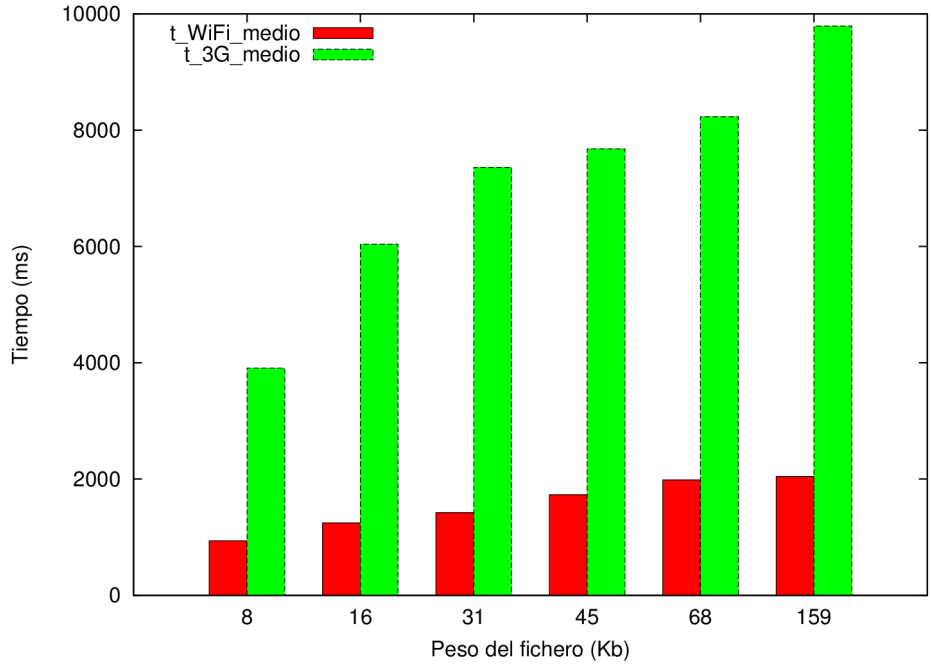


Figura 7.7. Tiempo de procesamiento en modo remoto incluyendo tiempo de envió y retorno del resultado

Estos tiempos son desde que hacemos la petición hasta que obtenemos la respuesta, es decir incluye toda la latencia de ida y vuelta en el intercambio de datos, para obtener los tiempos por separado de lo que cuesta enviar, procesar y el retorno del resultado hacemos una estimación de los tiempos basándonos en los valores que obtenemos con la herramienta de Linux *my traceroute* (mtr) que con mas de diez mil *pings* hacia el servidor de *face.com,* medimos el tiempo de ida y vuelta usando

peticiones ICMP, como lo podemos observar en la *figura 7.8* obteniendo en el ultimo salto un promedio de 105,8 ms con una desviación estándar de 17.8 ms y una tasa de perdida de 1.1%.

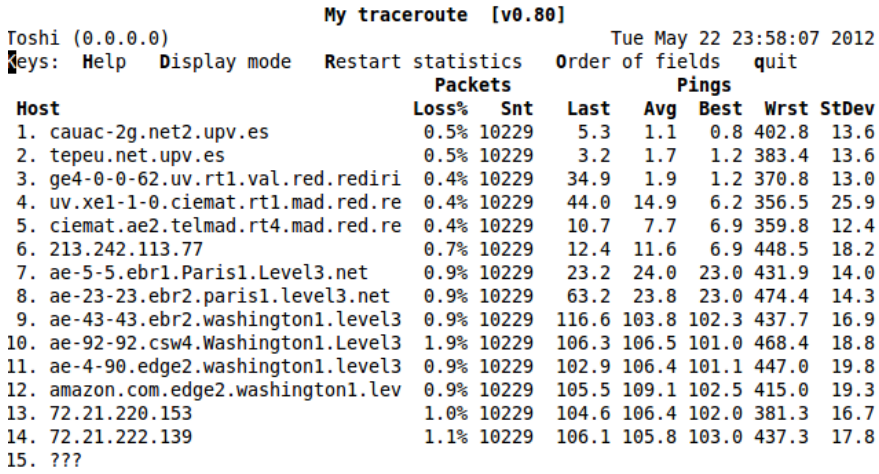


Figura 7.8. Herramienta MyTraceRoute de Linux que es como la suma de ping y traceroute dirigida hacia face.com

Lo que nos deja pensar que si el tiempo desde que enviamos la imagen y obtenemos la respuesta en una conexión WiFi es de dos segundos y el tiempo de ida y vuelta de un paquete sin confirmaciones es de 105ms tendríamos tiempos de procesamiento en modo remoto inferiores al segundo y es razonable por los potentes equipos que poseen ejecutando las aplicaciones de reconocimiento facial, e incluso pueden ser menores si sus algoritmos se están ejecutando sobre GPUs.

Así esta transferencia se realiza en el mejor de los casos en dos segundos con una conexión WiFi y de cuatro segundos con una conexión móvil 3G con ficheros muy pequeños

7.4. Pre procesado en local y procesado en remoto

La idea principal es: realizar desde el móvil la captura y la compresión de la imagen, luego enviársela al procesamiento en remoto, y el resultado que obtenemos lo desplegamos en pantalla, con esta consideración podemos realizar las siguientes combinaciones:

a) Capturar la imagen y enviarla tal cual.
b) Capturar, comprimir la imagen y enviarla.
c) Capturar, comprimir, convertir la imagen a escala de grises y enviarla.
d) Capturar, comprimir, convertirla a escala de grises, recortar de la imagen únicamente el rostro y enviarla.

Todos los procesos de comprimir la imagen se realizan mediante el estándar JPG y el tiempo que toma realizar este proceso junto a los otros como captura, convertir a escala de grises se realizan en razón de mili segundos y lo podemos observar en la *tabla 4*.

Para cada una de estas combinaciones y en base a las pruebas anteriores podemos decir que:

- para la primera combinación capturamos la imagen y la enviamos tal cual para su procesamiento en remoto y esperamos el resultado, que es el caso de utilizar la imagen de

tamaño de 160 kB en modo remoto, *face.com* cuando recibe imágenes con una elevada resolución las re dimensiona, pero en esta opción no estamos pre procesando nada.

- Para la segunda combinación comprimimos la imagen y la enviamos, que son los casos de utilizar imágenes de 16, 31, 45 y 68 kB en modo remoto, esta es la mejor opción porque no desperdiciamos el tiempo haciendo el resto de pre procesos y aprovechamos la compresión de la imagen para conseguir tiempos menores en la transmisión de la misma.

- El tercer caso consiste el anterior mas los 600 ms que cuesta convertir la imagen en escala de grises para enviar únicamente imágenes de 12 kB, estos 600 ms no se compensan con el tiempo que se ahorra en transferir una imagen de este tamaño.

- Y la ultima combinación consiste en todos los tres casos anteriores mas 12 ms que cuesta recortar el rostro de la imagen y podemos transferir imágenes 7 u 8 kB, pero esto acarrea el problema que el porcentaje de confiabilidad desciende a un 81%.

Para la funcionalidad en general de la aplicación se ha decidido realizar tareas complementarias o cooprocesos, así en local obtenemos el nombre de la persona que previamente hayamos entrenado y con la imagen comprimida de 14kB ± 2kB que enviemos a remoto obtenemos el reconocimiento de género, el estado de ánimo y la estimación de la edad.

Face.com para el reconocimiento facial se integra con las redes sociales para obtener los nombres de las personas etiquetadas en las fotografías que tenemos en una cuenta de *facebook* o *Twitter*, enviando las credenciales de estas cuentas mediante el protocolo Oauth e interactuando con otras APIs. Esta funcionalidad no está implementada actualmente en la aplicación.

El último de los sub procesos que se lleva a cabo es el tratamiento del resultado en el que también son muy pocos los mili segundos que se necesitan para *parsear* la respuesta y mostrarla en pantalla.

7.5. Reproducción de condiciones de la red

Para garantizar rendimientos, anchos de banda y bajas latencias el núcleo de Linux incorpora herramientas de encaminamiento, filtrado y clasificación de paquetes, por medio de las utilidades y herramientas agrupadas en *iproute2* entre las principales podemos nombrar a: *arpd*, *cstat*, *ifcfg*, *ip* y *tc,* estas dos ultimas forman *LARTC Linux Advanced Routing and Traffic Control* [26] para gestionar el tráfico que pasa por su interfaz hacia la red.

La herramienta *tc* mediante la configuración de colas y sus disciplinas consigue establecer políticas de clasificación, marcado, descarte, retraso y re envió de paquetes. estas normas de control se aplican sobre dos tipos de colas: la cola *ingress* y la cola *egress*.

En la cola *ingress* se realiza una etapa de tratamiento de paquetes en la cual se puede descartar y/o marcar y la cola *egress* adopta una estructura de árbol con filtros para distribuir los paquetes entre las clases y las colas. Para realizar estas tareas, *tc* posee elementos como: clases, disciplinas, filtros y marcadores.

elemento	función
class	*shaping*
qdisc	*schedulling*
filters	*clasifier, policer, dropper*
markers	*marker*

Mediante las clases se realiza una planificación de paquetes. Con las disciplinas se ordenan los paquetes antes de desencolarlos, la disciplina por defecto es una simple cola FIFO. Los filtros se utilizan para clasificar, separar y limitar el tráfico en una cola en concreto, también controla que un elemento no consuma más de lo que debe y además descarta paquetes cuando exceden el ancho de banda establecido.

Los paquetes son encolados en la interfaz de salida y es donde se define el control de tráfico y si los paquetes son encolados o si son descartados, la disciplina de colas básica en Linux se muestra en la *figura 7.9*

Para añadir escenarios con retardos y perdidas de paquetes necesitamos de un emulador de red que conecte varios sistemas finales para ello hacemos uso de NetEm [27]. que permite tomar la red de área local perfecta y convertirla en una red lenta y pesada, perfecto para evaluar en nuestro ordenador protocolos y aplicaciones que tienen que ejecutarse en redes de área extendida.

Ethernet tiene una latencia de 100 micro segundos y puede transferir 100 megabits por segundo, en una conexión de banda ancha están disponibles una amplia variedad de velocidades desde 128k a 4Mbits pero este tipo de conexión acarrea el problema de la latencia que generalmente rodea los 50 mili segundos.

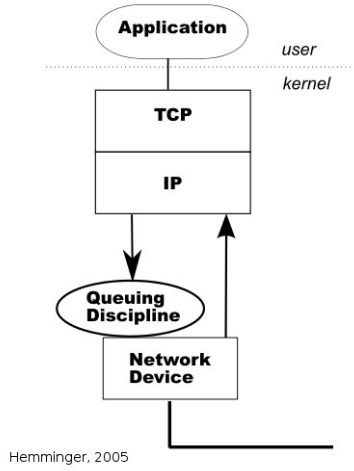


Figura 7.9. Disciplina de colas Linux

Internamente NetEm es un esquema de disciplina de colas que consta de dos colas de paquetes, la primera es una cola de espera y la segunda es una cola anidada. Cuando la interfaz encola un paquete, se toma el paquete se le pone una marca de tiempo y un tiempo de envió y se coloca en la cola de espera, un *timer* mueve los paquetes de la cola de espera a la cola anidada de donde la interfaz los coge para desencolarlos.

Originalmente NetEm se comporta como una cola FIFO sin retardos ni perdidas, para modificar estos parámetros lo hacemos mediante el comando *tc*.

El primer parámetro que podemos modificar es el retardo, considerando que las redes no tienen un retardo constante sino variante dependiendo del tráfico que fluye en la misma red, por lo cual podemos asignarle una distribución normal uniforme de un valor medio mas/menos la desviación estándar.

El segundo parámetro es la perdida de paquetes que lo especificamos en porcentaje, NetEm elimina al azar tantos paquetes sean necesarios para ajustarse a este porcentaje antes de encolados.

A continuación se muestra la configuración de los parámetros con datos obtenidos en el punto 7.3

```
netem  delay 106ms 18ms distribution normal loss 1.1%
```

Existen otros parámetros que también se pueden configurar entre ellos: duplicación, re ordenamiento pero en estas pruebas no los hemos utilizado.

Nuestro caso de prueba es emular la petición POST a los servidores de *face.com*, para ello tenemos una maquina virtual ejecutando un servidor web y desde el sistema que aloja al sistema virtual lanzamos el conjunto de pruebas variando los parámetros de NetEm para obtener los resultados.

Para emular el tiempo los procesos de envió, procesado y retorno variamos el retardo con valores de 2, 10, 50, 100 y 500 mili segundos y para cada uno de estos retardos variamos la perdida de paquetes desde 0% de perdida es decir un canal perfecto e ideal a 1, 5, 10 y 20%, este proceso lo repetimos una decena de veces, obteniendo los valores que están representados en la *figura 7.10*.

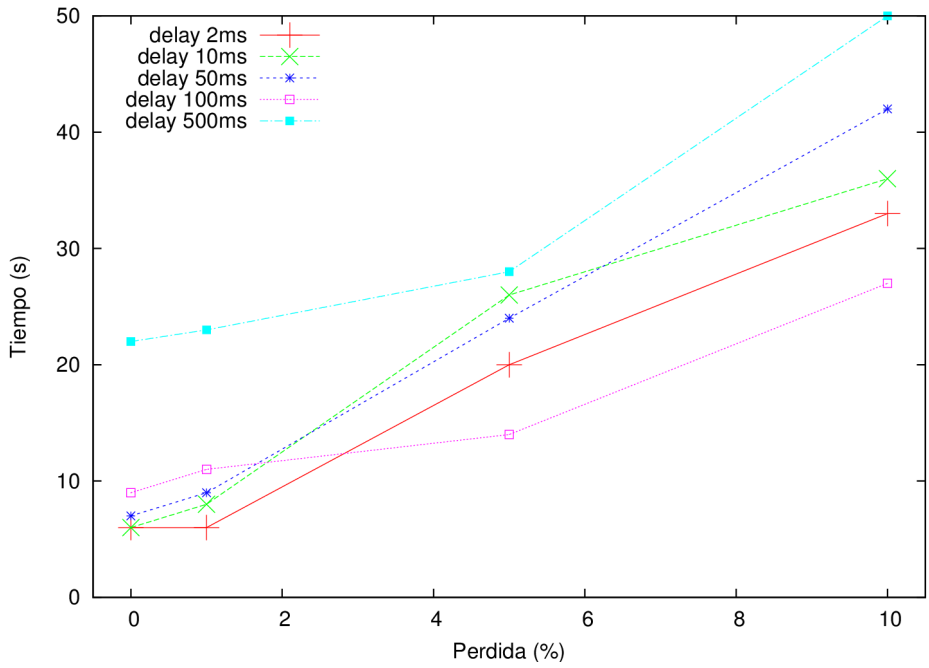


Figura 7.10. Tiempo obtenidos variando la tasa de error y el retardo de la conexión 3G

Podemos apreciar que con el canal ideal el tiempo en hacer una petición y obtener la respuesta es de seis segundos con el mínimo retardo y 22 segundos para el retardo máximo, luego se puede destacar que el retardo de 100 mili segundos en la perdida de 5% de los paquetes tiene el valor mínimo de cerca de 14 segundos y estos valores se disparan conforme crece la tasa de errores.

Por ultimo podemos obtener la latencia media: en el caso que fijamos la latencia base en 2 mili segundos obtenemos una latencia de cerca de 800 mili segundos. Y el caso extremo en que la latencia obtenida llega a ser de 1800 mili segundos se debe a cuando fijamos en la base en 500, como lo podemos observar en la *figura 7.11.*

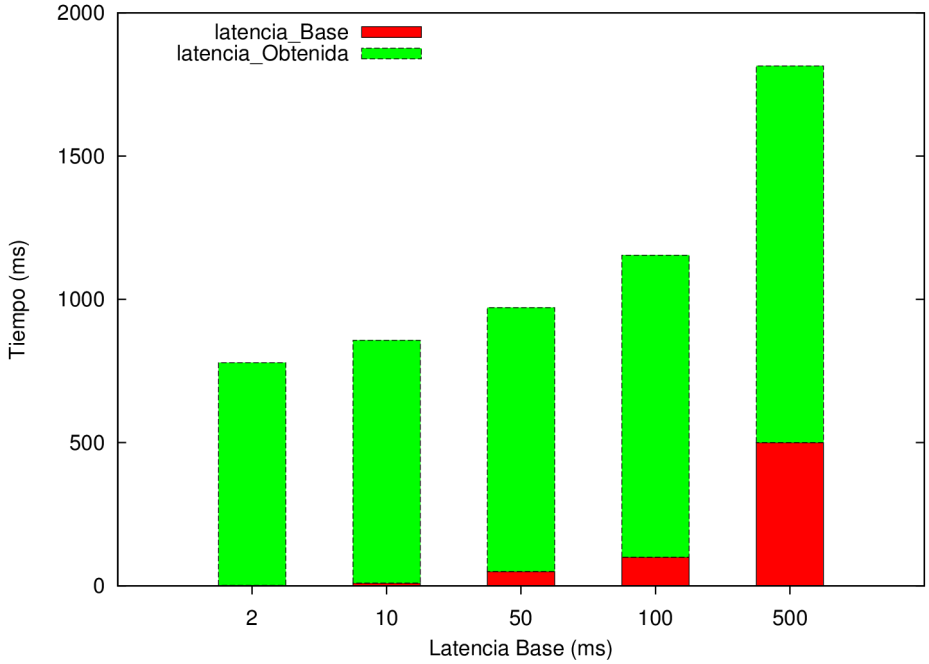


Figura 7.11. Tiempos de latencia obtenidos con NetEm

7.6. Ahorro de energía

Para este apartado nos basamos en el análisis para determinar cuando resulta beneficioso descargar la computación en el *cloud computing* de coste/beneficio de Kumar [5].

Suponemos que la computación requiere C instrucciones. S y M son la velocidad de instrucciones por segundo en el servidor en la nube y en el dispositivo móvil respectivamente, realizar la misma tarea C en el servidor es $\frac{C}{S}$ y en el móvil seria $\frac{C}{M}$.

El servidor en la nube y el dispositivo móvil intercambian D bytes de datos sobre un ancho de banda de red B , transmitir y recibir estos datos toma $\frac{D}{B}$ segundos.

Mediante los parámetros *power consuming* P_c , *power idle* P_i , *power transmition* P_{tr} que son específicos para cada dispositivo móvil, por ejemplo con el teléfono *Samsung Galaxy Ace* con procesador *ARM versión* 6 de 800-MHz tenemos M=800 y:

$P_c \approx 0{,}9\,W$
$P_i \approx 0{,}3\,W$
$P_{tr} \approx 1{,}3\,W$

Si el sistema móvil realiza la computación el consumo de energía es $P_c * (\frac{C}{M})$ (1)

Si el servidor realiza la computación la energía consumida es $P_i * (\frac{C}{S}) + P_{tr} * (\frac{D}{B})$, (2)
entonces la cantidad de energía que se ahorra cuando el móvil descarga el computo en el servidor es:

$$P_c * (\frac{C}{M}) - P_i * (\frac{C}{S}) - P_{tr} * (\frac{D}{B}) \quad (3)$$

sabemos que el servidor es F veces más rápido que el móvil $S = F * M$, remplazamos en (3) obtenemos la ecuación del consumo de energía:

$$\frac{C}{M} * (P_c - \frac{P_i}{F}) - P_{tr} * (\frac{D}{B}) \quad (4)$$

Esta formula produce un valor positivo cuando $\frac{D}{B}$ es suficientemente pequeño comparando con $\frac{C}{M}$ y F que deben ser suficientemente grandes.

En [11] nos dice que en *face.com* utilizan un servidor con 8 cores y suponemos una frecuencia de reloj de 3.2-GHz, así la aceleración F esta dada por:

$$\frac{S}{M} \approx \left[\frac{(3{,}2 * 1024 * 8 * x)}{800} \right]$$

donde: x es la aceleración dada por otros recursos adicionales como la memoria o el *pipelining* más agresivo, entre otros. Asumimos que x=5 y obtenemos que $P_c \approx 160$, el servidor es 160 veces más rápido que el dispositivo móvil y este valor se puede incrementar si la computación en la nube es paralelizada y repartida entre múltiples cores. Si asumimos que F=160, reemplazando estos valores en la ecuación de la energía obtenemos:

$$\frac{C}{800} * (0{,}9 - \frac{0{,}3}{160}) - 1.3 * (\frac{D}{B}) \approx (0{,}0011226 * C) - 1.3 * (\frac{D}{B})$$

igualamos la ecuación a cero y tenemos:

$$B_0 = 1158{,}03 * (\frac{D}{C}) \tag{5}$$

donde B_0 es el mínimo ancho de banda requerido para ahorrar energía determinada por el ratio entre *D/C,* si este valor es bajo la descarga es benéfica y puede ahorrar energía, debido a que se cumple el principio de realizar grandes cantidades de computación C con relativamente pequeñas cantidades de comunicación D .

El enorme potencial que ofrece el *cloud computing* para ahorrar energía esta comprometido con temas de seguridad, privacidad, confiabilidad y manejo de datos en tiempo real, esto puede significar el uso de técnicas de encriptación de datos antes de enviarlos a la nube y esto requiere un procesamiento adicional C_p en el dispositivo móvil y añade un parámetro a la ecuación del consumo de energía y quedaría como:

$$\frac{C}{M} * (P_c - \frac{P_i}{F}) - P_{tr} * (\frac{D}{B}) - P_c * (\frac{C_p}{M}) \tag{6}$$

donde $P_c * (\frac{C_p}{M})$ es la energía requerida para asegurar confidencialidad y privacidad de la información. Si este valor es significativo la descarga de computación hacia *cloud computing* no ahorra energía en el móvil.

7.7. Resultados

Los valores que a continuación se muestran son valores medios normalizados y han sido obtenidos en la ejecución de la aplicación de principio a fin, esta ejecución se ha repetido una decena de veces para que los datos sean más confiables, centrándonos en los tiempos de respuesta y precisión en cada uno de los modos.

proceso	**Tiempo (ms)**					
	local			**remoto**		
captura	480			480		
comprimir	180			180		
escala de grises	595			-		
detectar rostro	667			-		
recortar	12			-		
re dimensionar	30			-		
ecualizar	10			-		
envío	10			Depende de la red y del tamaño de la imagen		
procesamiento	Depende del número de imágenes [*tabla 5*] y [*tabla 6*]			≈ 600		
retorno	10			Depende de la red [*tabla 7*]		
resultados	142			113		
total	min	2137	2950	min	1712	4681
	max	252136	4181	max	2818	10563
		***T**	****R**		**WiFi**	**3G**

Tabla 4. Tiempos obtenidos en la ejecución de todos los procesos

*T= Entrenamiento
**R=Reconocimiento
- = no se realiza

Como podemos ver en la *tabla 4* tenemos reconocimientos entre 3 y 4 segundos desde que presionamos el botón reconocer hasta obtener el resultado en modo local y la misma petición en modo remoto requiere de 2 a 3 segundos aproximadamente en una red WiFi o de 5 a 10 segundos sobre una conexión 3G.

Tabla 5.) Tiempo para el entrenamiento de _*i*_ imágenes:

Imágenes	Tiempo (s)
5	1
10	3
15	6
20	11
25	17
30	23
35	30
40	40
45	56
50	63
55	76
60	87
65	103
70	124
75	148
80	152
85	178
90	198
95	221
100	250

Tabla 6.) Tiempo para el reconocimiento entre _*i*_ imágenes:

Imágenes	Tiempo (ms)
5	814
10	930
20	966
30	1036
40	1040
50	1165
60	1390
80	1622
100	1812

Los tiempos de respuesta con las imágenes originales se encuentran alrededor de 2045 mili segundos de promedio utilizando la red WiFi y 9800 mili segundos con la red 3G.

	Tiempo (ms)	
Peso (kB)	WiFi	3G
8	939	3908
16	1245	6038
31	1424	7360
45	1730	7680
68	1985	8233
159	2045	9790

Tabla 7.) Procesamiento remoto mediante el envió y recepción en la red _*x*_ con imágenes de tamaño _*i*_

En la *tabla 7* observamos que los tiempos de respuesta se decrementan conforme baja la calidad de imagen y la creación de versiones comprimidas de las imágenes que van a ser enviadas al servidor en lugar de las originales es de solo 180 mili segundos [*tabla 4*]. Lo que nos indica que necesariamente debemos realizar la compresión a niveles moderados donde la calidad de la imagen no se vea afectada con el decremento del nivel de resolución de la misma, al fin y al cabo los valores de confianza obtenidos con las diferentes resoluciones son altos y están sobre el 85% [*tabla 9*] entre la versión de la imagen de 16 kB frente a la misma en su versión original de 159 kB.

	numero de imágenes por persona				
personas	**1**	**2**	**3**	**4**	**5**
5	0	41	63	77	81
10	15	51	68	75	79
20	46	70	77	84	87

Tabla 8.) Valores medios porcentuales de confianza obtenidos en reconocimientos en modo local

Peso (kB)	confianza
8	81
16	86
31	86
45	87
68	89
159	93

Tabla 9.) Valores medios porcentuales de confianza obtenidos en reconocimientos en modo remoto.

Dado que los datos que se intercambian son pequeños ficheros no se requiere un gran ancho de banda para transmitirlos, pero el tiempo de respuesta de *face.com* puede jugar un papel muy importante en el tiempo de transmisión total, sin embargo estos resultados conseguidos pueden ser considerados como indicadores que reflejan un experimento sobre un escenario real.

Capítulo 8. Conclusiones

En este trabajo hemos realizado un estudio de como la descarga de computo beneficia a una aplicación móvil en concreto haciendo uso de software como servicio SaaS del paradigma *cloud computing,*

La decisión de si el procesamiento lo hacemos en local o lo hacemos en remoto la tomamos basándonos en la cantidad de computación y comunicación que realiza la aplicación siempre con la premisa de **descargar el procesamiento si realizamos poca comunicación y mucha computación**. Y en casos de comunicación y computación en partes iguales depende del canal sobre el cual se efectúe la comunicación.

La aplicación elegida fue de reconocimiento facial con la idea intuitiva de que los cálculos matemáticos exhaustivos que esta requiere para correlacionar los datos de entrada ponen en compromiso las características hardware del dispositivo móvil. Mientras que si esta misma aplicación es ejecutada en otros sistemas como un servidor, se realiza con mucho menos esfuerzo y en mucho menos tiempo.

El tema del tratamiento de imágenes y de identificar el contenido visual de las mismas de por si es interesante, actual y con futuro, pero requiere tener ciertas nociones de técnicas ampliamente usadas en otros campos como la inteligencia artificial y la visión por computador.

Para realizar automáticamente reconocimientos es necesario una etapa de entrenamiento o aprendizaje de lo que queremos que reconozca la aplicación en nuestro caso son rostros, siendo esta etapa la que más tiempo requiere y la que en realidad pone a prueba los recursos hardware del sistema. Esta etapa está directamente relacionada con el número de imágenes de las que tiene que aprender, **mientras más muestras de un caso se posea más precisos son los resultados**, creando un primer *trade-off* entre el tiempo requerido en el entrenamiento y el número de muestras a entrenar.

El reconocimiento en si no conlleva mucho tiempo para su ejecución porque únicamente realiza una sola vez la extracción de las características para conseguir los valores de los coeficientes de correlación a ser comparados con cada uno de los coeficientes de la matriz obtenida en el entrenamiento, el resultado del reconocimiento es la mínima diferencia de la comparación de estos coeficientes.

Cuando delegamos el procesamiento a entidades externas nos beneficiamos de la descarga de computo en términos de utilización de recursos hardware inclusive en ahorro de la batería pero tenemos el problema de la latencia en la comunicación con esta entidad externa, para reducir un poco este problema se realiza un pre procesamiento de la información a ser enviada, en lugar de enviar una imagen de millones de pixels, enviamos una versión comprimida de la misma con un nivel de compresión del 80% mediante el estándar JPG, de forma tal de que la calidad de la imagen no se vea afectada para la extracción de su información.

En los resultados basándonos en el nivel de confianza medio de un 86% obtenido con el envió de imágenes de 16 kB se puede decir que es más que aceptable el reconocimiento, dependendiendo eso si del contexto en que usemos el resultado, aquí surge un segundo *trade-off* entre el tamaño de imagen y el tiempo que cuesta transmitir esta imagen.

Con los mismos parámetros: la imagen de 16 kB y la conexión WiFi, los resultados del reconocimiento que se obtiene son tiempos inferiores a cuando son realizados en forma local incluso contando con la latencia en la comunicación de ida y vuelta. Llegando a una segunda conclusión: que en este caso en concreto **el procesamiento remoto es más rápido y más fiable en términos de precisión de los resultados**.

Por otro lado podemos decir que para delegar el procesamiento a entidades externas en muchos de los casos se sigue la filosofía de pago por uso, en donde "tanto usas - tanto pagas", en este caso hemos encontrado un proveedor que de momento ofrece sus servicios de forma gratuita pero eso no quita que puedan cambiar de idea y seguir a otros proveedores que cobran por cada consulta realizada o ofrecen sus servicios mediante planes mensuales con un cupo limitado de consultas, a cambio de dar mejores prestaciones con prioridad en las peticiones y soporte técnico en casos de problemas.

Lo que nos deja otra conclusión que **el procesamiento en remoto en el caso de usar algún proveedor SaaS es más costoso desde el punto de vista del usuario quien tiene que pagar quizás por el uso de la aplicación y además por la línea de datos de su operador de telefonía**.

En el proceso de comunicación entre el cliente y el servidor los datos están expuestos a problemas de seguridad inclusive si están cifrados debido a que están expuestos a ser esnifados por cualquier persona/maquina que este escuchando el canal de comunicación lo que nos deja otra conclusión que **el proceso de comunicación para transferir la información para su procesamiento en remoto es menos fiable que si simplemente lo realizamos en local** donde no necesitamos transferir datos.

Finalmente podemos decir que los beneficios de SaaS están a la vista desde tres puntos de vista:

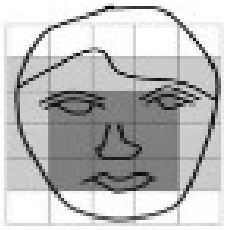
- Desde el punto de vista del usuario, SaaS evita la compra de licencias de software que quizás no lo usemos.
- Desde el punto de vista del desarrollador la decisión de tercerizar el procesamiento de información donde por un módico precio puede obtener más prestaciones por parte de los proveedores de los muchos que abiertamente invitan a consumir sus servicios por medio de sus APIs, puede elegir siempre la que menos tiempo le lleve en adaptarse y le permita seguir empleando sus técnicas y modos de trabajo a los que está acostumbrado.
- Y desde el punto de vista del proveedor de servicios, SaaS les permite ofrecer cada vez mejores servicios de forma totalmente escalable con la demanda que se le presente y algo muy importante es que evitan la piratería de software.

Con la constante evolución de los dispositivos móviles para futuros esfuerzos se puede ofrecer aplicaciones que combinen estos dos modos de operación, con el objetivo de obtener resultados en menos de un segundo en reconocimientos locales con un vector de entrenamiento previamente cargado y si el resultado no se puede conseguir en local o es desconfiable enviar la consulta a modo remoto obteniendo la respuesta de entre dos a cinco segundos y si esta respuesta también es desconfiable o tampoco se encuentra entre las billones de imágenes que pueden manejar los servidores, ofrecer la opción de agregarlas como un nuevo registro para futuras consultas al estilo *crowdsourcing*. Ya no en el campo de reconocimiento facial por la privacidad de la identidad de las personas que debemos manejar pero si por ejemplo con objetos o escenas que se puedan identificar en una imagen.

Capítulo 9. Anexos

9.1. Anexo 1: Guía de uso

FaceRec



Reconocimiento Facial Móvil

Dispositivos

Se ha escogido trabajar con la plataforma Android por las capacidades multimedia, accesibilidad, facilidad de interacción y conectividad inalámbrica que nos ofrece a muy bajo coste.

Requisitos

Hardware

Las librerías en C de la aplicación *FaceRec* están compiladas para dispositivos con procesadores de arquitectura ARMv6.

Presente en muchos dispositivos lanzados a partir del año 2011.

Software

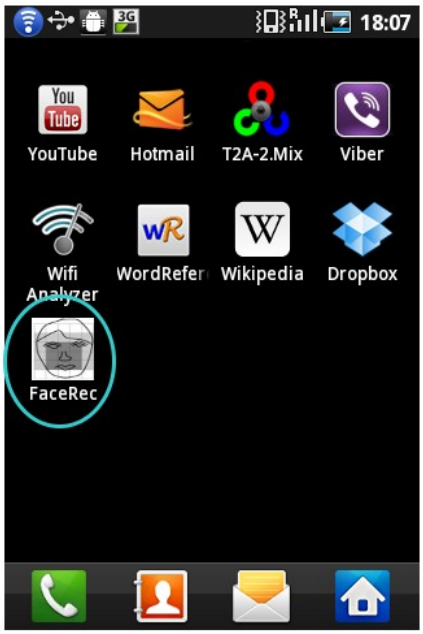
La versión mínima de sistema operativo requerido es Android Froyo (2.2).

Instalación

Actualmente por medio de la distribución del fichero LocalRemote.apk, pulsamos sobre el mismo, aceptamos los permisos requeridos por la aplicación, se inicia la instalación y el proceso termina muy rápidamente con la opción de lanzarla a ejecución directamente.

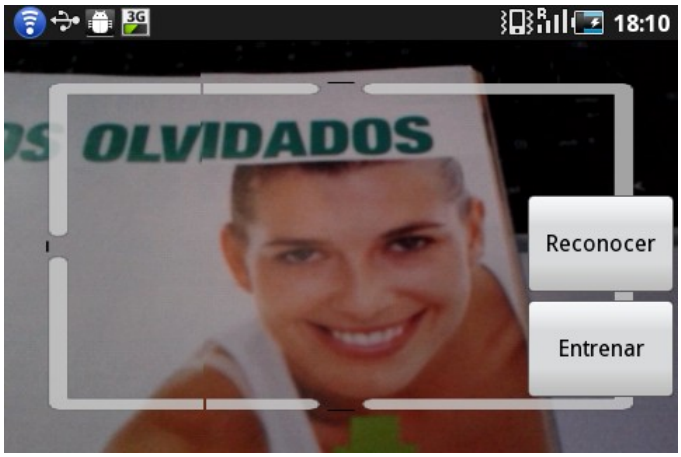
Inicio de la aplicación

Para ejecutar la aplicación de reconocimiento facial móvil pulsamos sobre el icono FaceRec.



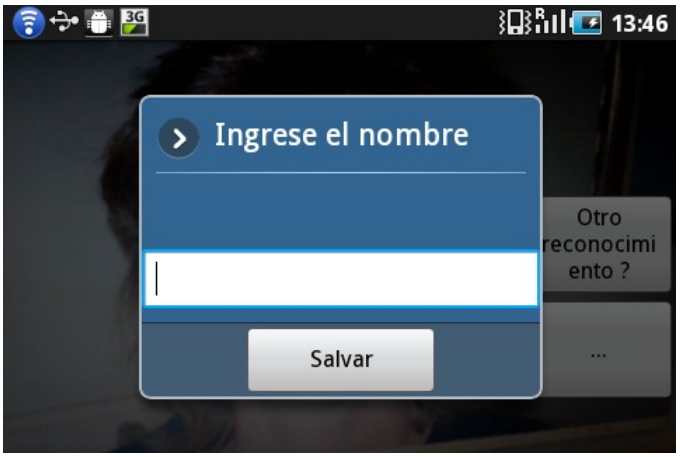
Interfaz de usuario

Se inicia la aplicación como si hubiésemos lanzado la aplicación cámara de fotos es decir se abre la cámara y se trabaja en modo horizontal con un punto de mira para enfocar algún rostro y se muestran dos botones “reconocer” y “entrenar”



Pulsando entrenar

Se inicia un *timer* para capturar automáticamente cinco fotografiás del rostro de una persona con un espacio de cinco segundos entre cada disparo, al final de este proceso nos solicita el nombre de la persona que hemos hecho las fotografías para identificarla dentro de la aplicación.



Pulsando reconocer

Se inicia el timer de cinco segundos para capturar una fotografía, tiempo en el cual el usuario puede enfocar mejor el rostro de la persona que se desea buscar dentro de la base de datos.

Como resultado del procesamiento local en la parte inferior de la pantalla se muestra el nombre de la persona encontrada, además se pinta un recuadro sobre su rostro.

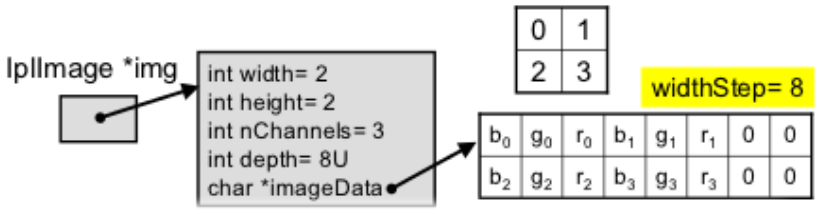
Si se posee conexión a Internet, se recibe el resultado del procesamiento remoto el cual es mostrado también en pantalla.

Dependiendo de la captura realizada el procesamiento en remoto nos devuelve o no la estimación de la edad.

9.2. Anexo 2: Principales tipos de datos y funciones para trabajar con OpenCV

El principal objeto con el cual trabajamos es el que representa las imágenes, dentro de esta librería se llama:
IplImage

Las imágenes siempre se descomprimen en memoria como una matriz de pixels,
Las variables se manejan con punteros a IplImage.
Ademas de imágenes la librería permite manejar matrices cvMat
el puntero *img->data* apunta a una matriz.



Los principales atributos de la imagen son:

size: tamaño
width: anchura
heigth: altura
depth: profundidad en pixels 8,16,32 bits
nChannels: numero de canales desde 1 hasta 4. Las imágenes en escala de grises tienen un sólo canal, mientras que las de color tienen 3 o 4 canales.

Tipos de dato utilizados:
IPL_DEPTH_8U - unsigned 8-bit integers para el preprocesado
IPL_DEPTH_32F - single precision floating-point numbers para las imágenes resultantes del eigen.

Por medio de la librería <highgui.h> tenemos entrada y salida de imágenes a disco, es decir podemos abrir y guardar imágenes.

Todas las imágenes después de ser utilizadas deben ser liberadas.

Principales funciones:

Acción	Función en OpenCV
Abrir imagen	img=cvLoadImage(fileName,flag); Flag > 0 para 3 canales Flag = 0 para 1 canal Flag < 0 para todos los canales
Guardar imagen	cvSaveImage(outFileName,img) OutFileName = nombre de fichero de salida Img = imagen que se va a salvar
Crear imagen	IplImage* cvCreateImage(CvSize size, int depth, int channels) Size = el tamaño Depth = la profundidad del pixel en bits Channels = numero de canales
Clonar imagen	IplImage* cvCloneImage (IplImage* entrada) Entrada = imagen a clonar
Copiar imagen	cvCopy(img, imageTmp) Img = imagen destino ImageTmp = imagen fuente
Liberar memoria	cvReleaseImage(&img); &img = liberar la imagen al dejar de utilizarla

9.3. Anexo 3: Imágenes eigen

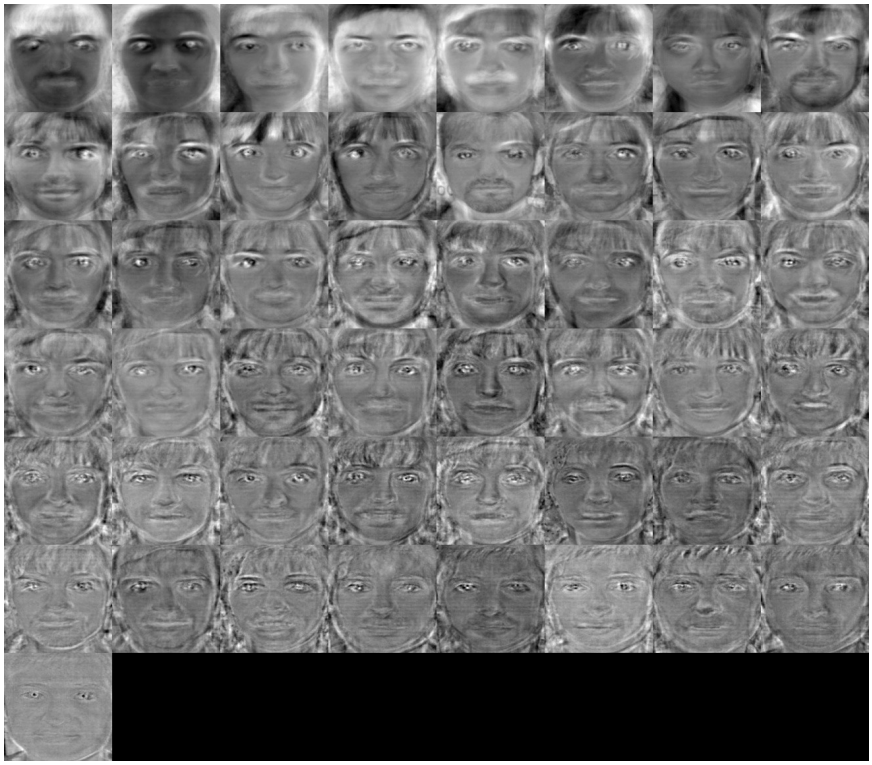
Datos obtenidos en un entrenamiento de 10 personas con cinco imágenes de cada una de ellas.
Imagen promedio representa a todas las imágenes de que se encuentran en el espacio eigen.
Imagen 0: es la que tiene los coeficientes representativos más altos de todo el espacio eigen.
Imagen 43: el la que tienes los coeficientes mas débiles, prácticamente es solo ruido.

Imagen promedio

eigenface 0

eigenface 43

Espacio eigen formado por diez personas con cinco fotografías de cada una.



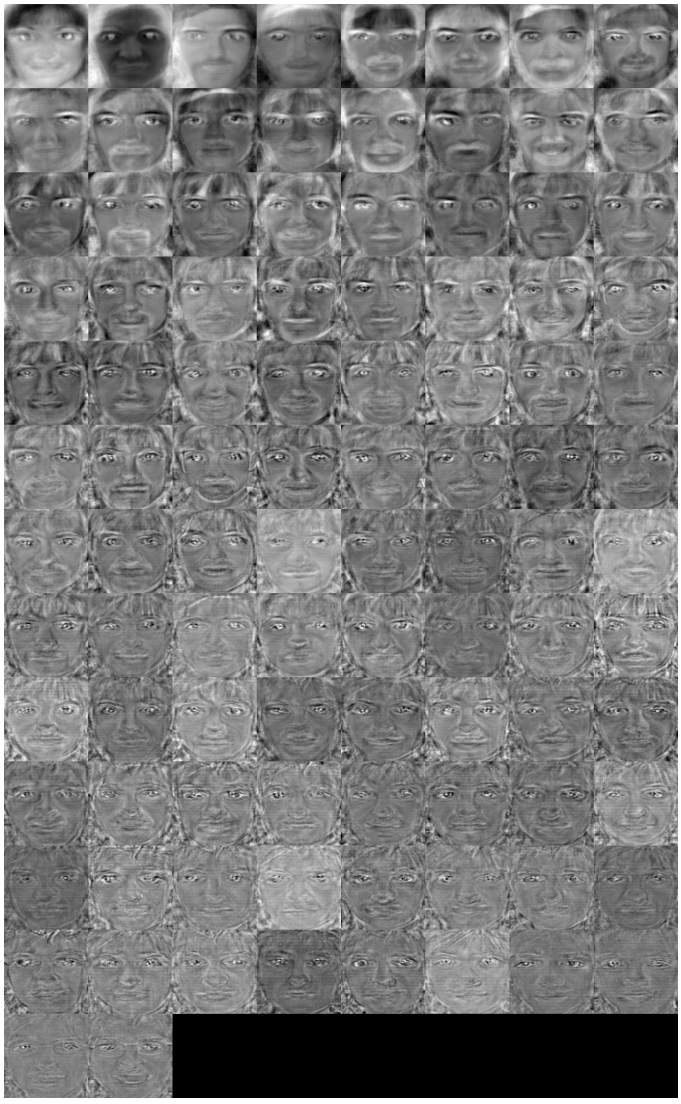
Datos obtenidos en un entrenamiento de 20 personas con cinco imágenes de cada una de ellas.
Imagen promedio representa a todas las imágenes de que se encuentran en el espacio eigen.
Imagen 0: es la que tiene los coeficientes representativos más altos de todo el espacio eigen.
Imagen 98: el la que tiene los coeficientes mas débiles, prácticamente es solo ruido.

Imagen promedio

eigenface 0

eigenface 98

Espacio eigen formado por veinte personas con cinco fotografías de cada una.



Capítulo 10. Bibliografía

Papers:

[1] Augmented Smartphone Applications Through Clone Cloud Execution
B. Chun, P. Maniatis
Intel Research Berkeley
HotOS 2009

[2] AlfredO: An Architecture for Flexible Interaction with Electronic Devices
Rellermeyer, Riva, Alonso
Systems Group, Department of Computer Science, ETH Zurich
Switzerland
Middleware 2008, LNCS 5346, pp. 22–41, 2008.
IFIP International Federation for Information Processing 2008

[3]The Case for VM-Based Cloudlets in Mobile Computing
M. Satyanarayanan, P. Bahl, R. Cáceres, N. Davies
Carnegie Mellon University
Published by the IEEE CS n 1536-1268
2009 IEEE

[4] Hyrax: Cloud Computing on Mobile Devices using MapReduce
E. Marinelli
September 2009

[5] Cloud computing for mobile users: can offloading computation save energy?
Kumar and Yung
Pardue University
Published by the IEEE Computer Society
April 2010

[6] Representación de caras mediante Eigenfaces
Lorente Gimenez Luis
Buran No. 11, mayo 1998

[7] Rapid object detection using a boosted cascade of simple features.
P. Viola and M.J. Jones.
CVPR 2001,

[8] An extended set of Haar-like features for rapid object detection.
R. Lienhart and J. Maydt.
ICIP 2002,

[9] Face Recognition: A Literature Survey.
W. Zhao, R. Chellappa, P J. Phillips, and A. Rosenfeld.
ACM Computing Surveys
2003

[10] M. A. Turk and A. P Pentland.
Face recognition Using Eigenfaces.
IEEE Conference on Computer Vision and Pattern Recognition
1991.

[11] Leveraging Billions of Faces to Overcome Performance Barriers in Unconstrained Face Recognition
Y. Taigman and L. Wolf
face.com
2011

[12] Efficient focusing and face detection. Face Recognition: From Theory to Applications,
Y. Amit, D. Geman, and B. Jedynak.
1998.

[13] Open Source Computer Vision Library - Reference Manual
1999-2001 Intel Corporation
U.S.A.

[14] Fun with Computer Vision: Face Recognition with OpenCV on the iPhone
Leon Palm
Chapter 5
Washington, DC

URLs : (fecha del ultimo acceso: 12 junio 2012**)**

[15] Mobile Cloud Computing: Issues and Risks from a Security Privacy Perspective: An analysis and a survey for my talk at secure cloud conference
March 5, 2010 By ajit
Open Gardens
http://www.opengardensblog.futuretext.com/archives/2010/03/mobile_cloud_co_2.html

[16] Vida de la batería preocupa a los usuarios de móviles
23 Sept. 2005;
http://edition.cnn.com/2005/TECH/ptech/09/22/phone.study/

[17] Desarrollo en Android
http://developer.android.com/sdk/ndk/index.html

[18] Face Detection using OpenCV
http://opencv.willowgarage.com/wiki/FaceDetection

[19] Indice de funciones de OpenCV
http://opencv.willowgarage.com/documentation/genindex.html

[20] Eigen faces
http://es.wikipedia.org/wiki/Vector_propio_y_valor_propio

[21] Distancia Mahalanobis
http://www.uv.es/ceaces/multivari/cluster/d_mahalanobis.htm

[22] API de face.com
http://developers.face.com

[23] Bases de datos para reconocimiento facial
http://www.face-rec.org/databases/

[24] Base de datos de caltech
http://www.vision.caltech.edu/html-files/archive.html

[25] Servicios Web de Amazon
http://aws.amazon.com/es/free/terms

[26] Linux Advanced Routing – Traffic Control
http://lartc.org/howto/

[27] NetEm
http://developer.osdl.org/shemminger/netem

[28] Especificaciones técnicas Samsung Galaxy Ace
http://www.samsung.com/galaxyace/ace_techspec.html