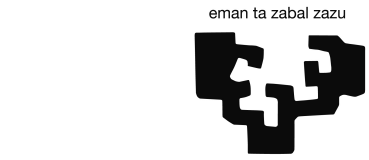


Universidad del País Vasco
Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Computación

Proyecto de Fin de Grado

Supervisión facial usando "deep features" desde un dispositivo Android

Autor

Jon Zabalza Peláez

2017

Resumen

El proyecto se basa en el reconocimiento facial en dispositivos Android mediante el uso de redes neuronales convolucionales. Para ello se utilizará MatConvNet para la extracción de las características de la imagen y posteriormente se investigarán ciertos métodos de clasificación entre los cuales se encuentra las máquinas de vectores de soporte (SVM). Una vez seleccionado el método adecuado se procederá a desarrollar una aplicación. A lo largo de este documento se detallará el funcionamiento de los métodos seleccionados, el proceso de creación de la aplicación y los resultados obtenidos.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

La telefonía móvil en los últimos años ha evolucionado de una forma vertiginosa, reduciendo el tamaño de los terminales y mejorando su rendimiento año a año, hasta tal punto que nuevas empresas son capaces de sacar al mercado una gran variedad de productos que llegan a competir con las grandes marcas. A la par que los teléfonos móviles, sus funcionalidades han ido aumentando ofreciendo nuevos servicios y mejorando los anteriores, convirtiéndose en elementos necesarios para el día a día.

Dado que los teléfonos móviles cada vez tienen mayor importancia y su uso trasciende del mero entretenimiento a un elemento cargado de información privada sus medidas de seguridad a su vez están mejorando para poder asegurar la confidencialidad de los datos. Así, la principal medida es el bloqueo del terminal mediante alguna técnica que valide la identidad del usuario.

Por ello en este proyecto se trata de investigar y desarrollar una aplicación de reconocimiento facial para verificar la identidad del propietario de forma fluida. Se investigaran diferentes métodos y se seleccionará el más adecuado para un dispositivo móvil de gama media. Para ello se utilizará la extensión MatConvNet [MCN, 2017] para MatLab a fin de obtener las características del rostro del usuario. A partir de estas se utilizará un clasificador binario para la verificación.

Finalmente, la aplicación usa las máquinas de vectores de soporte (SVM) como clasificador. Para ello, las imágenes se enviaran a Matlab desde el teléfono móvil para que este pueda obtener los vectores de características. Después, se enviaran de vuelta y el teléfono mediante las SVM los procesará y obtendrá el clasificador. En caso de realizar la

supervisión, será una única imagen la que se envíe al servidor y con su vector ya en el dispositivo, se obtendrá la respuesta binaria del clasificador. Por lo tanto, la comunicación entre Matlab y el teléfono móvil se lleva a cabo como en las figuras 1.1 y 1.2.

Además de esto, para comprobar la efectividad en la supervisión facial, se lleva a cabo una experimentación en la que se comparan los vectores de características obtenidos mediante MatconvNet, los histogramas convencionales y los histogramas de gradientes orientados.

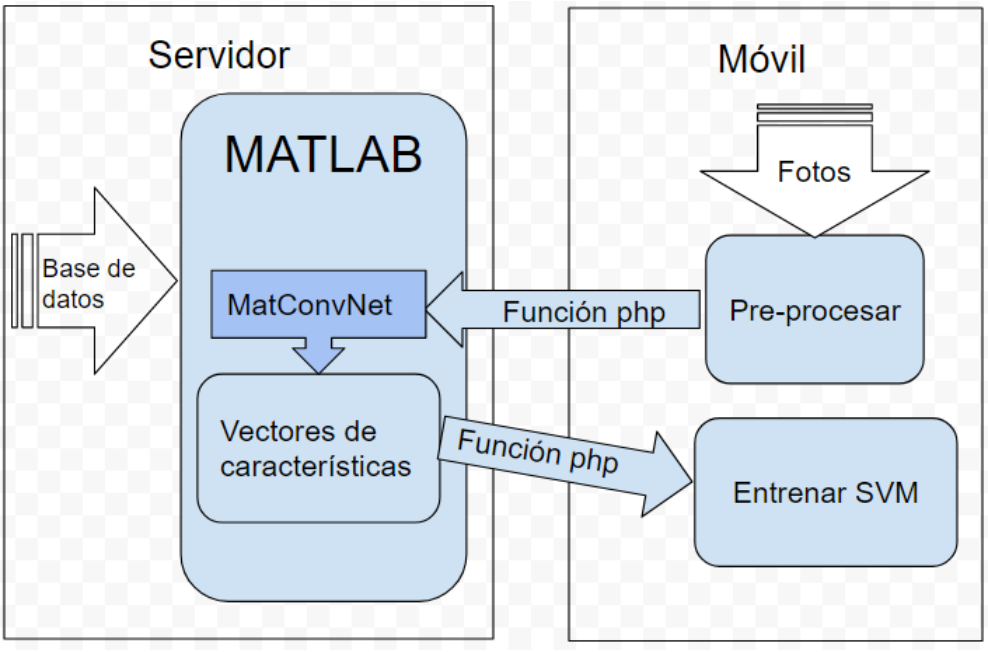


Figura 1.1: Generar clasificador

Esta memoria está estructurada en 6 diferentes capítulos sin contar con la introducción y los anexos. En los anexos A y B se documenta el uso de las herramientas de MatConvNet y LibSVM, en el anexo C se pueden ver los diagramas de secuencia de dos casos de uso y en los anexos D y E el código de la aplicación. Los capítulos son los siguientes:

- Documento de objetivos del proyecto: Estructuración de fases y tareas.
- Antecedentes: Documentación de las bases de la aplicación como son las redes neuronales y las máquinas de vectores de soporte.
- Experimentación: Exposición de los experimentos o pruebas realizados y sus resultados.
- Aplicación: Explicación del desarrollo de la aplicación.

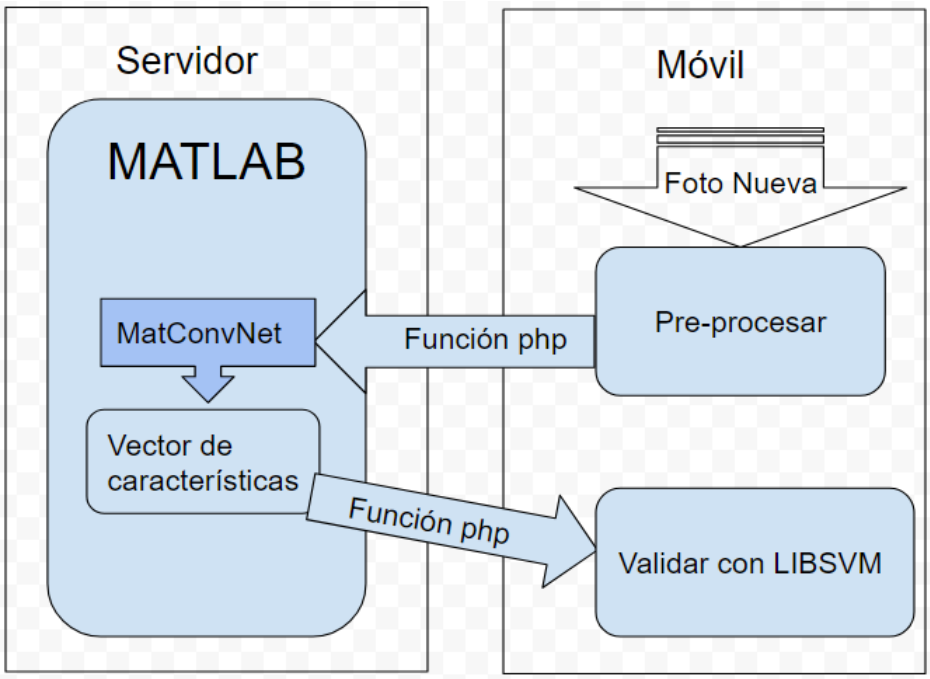


Figura 1.2: Verificar usuario

- Problemas: Problemas encontrados durante el proyecto y soluciones.
- Conclusiones y líneas futuras

2. CAPÍTULO

Documento de objetivos del proyecto

2.1. Fases del proyecto

El proyecto estará dividido en tres fases, en la primera de estas se pondrán a punto las aplicaciones necesarias para poder realizar las posteriores pruebas y además se seleccionarán las herramientas necesarias para la realización de la aplicación.

En la siguiente fase, se realizará un estudio del deep learning para el reconocimiento facial con redes neuronales y clasificadores lineales como las máquinas de vectores de soporte.

En la última fase, se desarrollará una aplicación en la que se usará el método más adecuado para mantener la minina comunicación entre MatLab y el teléfono móvil.

Durante el final de la tercera fase se decidió añadir una cuarta fase, para poder completar de manera adecuada esta memoria. En ella se hará un repaso a todos los capítulos y se añadirá todo lo que estaba previsto. Esto se explica en mayor profundidad en el capítulo 6.

2.1.1. Primera Fase

- Realizar la instalación de Matlab R2015a junto con la herramienta de Matconvnet para el uso de redes neuronales convolucionales e investigar cómo obtener el vector de características. Realizar una prueba para así poder realizar la posterior extracción de las características de la imagen.

- Realizar la instalación de Android Studio versión 2.2.3 y familiarizarse con su uso. Conforme avance el proyecto se irá actualizando a la versión más nueva. Esta herramienta facilita el desarrollo de aplicaciones Android y dispone de un emulador para testearlas. Además, realizar tutoriales para dedicar el tiempo de desarrollo plenamente a este y no tanto a la comprensión de la creación de aplicaciones básicas.

- Realizar la instalación de un editor de Latex [Lamport, 1986] y inspeccionar la plantilla proporcionada por la facultad para la realización de la memoria.

- Búsqueda y selección de aplicaciones similares de supervisión facial, base de datos y software adecuado para Android. Estas serán las bases de la aplicación para lograr ejecuciones más ágiles, adecuadas a la reducida capacidad de cómputo de la que dispondrá la aplicación, instalada en el teléfono, para poder realizar el reconocimiento.

2.1.2. Segunda Fase

- Elegir una muestra de fotos para llevar a cabo el pre-procesado para la posterior realización de unas pruebas iniciales.

- Investigar acerca de redes neuronales y máquinas de vectores de soporte, determinar los métodos a utilizar y documentarlo.

- Extraer los vectores de características de las imágenes pre-procesadas.

- Llevar a cabo la experimentación, comparando los vectores de características generados por técnicas de deep learning y los histogramas convencionales.

- Analizar los resultados obtenidos en la experimentación para poder determinar de forma preliminar cuál será el método más adecuado para la aplicación, manteniendo la mejor relación coste computacional/porcentaje de acierto debido a que la validación se llevará a cabo en el dispositivo.

2.1.3. Tercera Fase

- Completar el desarrollo de la aplicación que sea capaz de comunicarse con una instancia de MatLab para llevar a cabo el entrenamiento.

- Analizar los resultados obtenidos de la aplicación y determinar su viabilidad para su uso como parte de una aplicación móvil con más características.

2.1.4. Cuarta Fase

- Ampliar la experimentación con histogramas de gradientes orientados y realizar las modificaciones oportunas en los demás capítulos.

2.2. Planificación

Una vez decidido el proyecto este comienza el 7 de Enero de 2017 y se finalizará para el 20 de julio para así poder realizar alguna modificación de última hora si fuera necesario. Teniendo en cuenta que este proyecto se realiza en paralelo a las asignaturas del segundo cuatrimestre, se dedicarán una cantidad diferente de horas durante la primera y segunda fase. Durante los dos primeros, se dedicarán de media 3 horas diarias excluyendo los fines de semana. Después se dedicaran una media de 5 horas. Cabe destacar que durante periodos vacacionales, la media de horas de trabajo será algo más reducida.

Además, queda añadir que salvo que haya algún inconveniente todas las semanas se llevará a cabo una reunión para repasar el progreso del proyecto y consultar dudas. Estas reuniones serán los jueves de 11 a 12:30 en la facultad.

Debido a la prolongación del proyecto y a la adición de la cuarta fase la fecha de finalización del mismo cambia al 16 de octubre.

2.2.1. Gráfico Gantt

Debido a las dimensiones del gráfico, en las siguientes figuras se mostrarán las diferentes fases a lo largo del tiempo y los plazos en los que se trabajarán sus actividades. En un principio la duración de la tercera fase iba a ser mayor pero debido a una serie de inconvenientes en el segundo se redujo.

- Primera fase: Las dedicaciones de esta primera fase son un tanto caóticas debido a que una de las decisiones iniciales era determinar si la carga de computación con redes neuronales se iba a realizar en el dispositivo Android o en un servidor. El documento de objetivos dependía en gran medida de las decisiones previas.

Esta aparente falta de organización inicial se suple más adelante notoriamente.

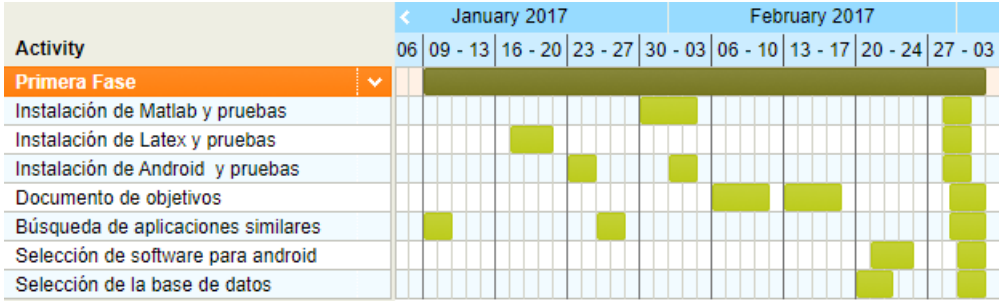


Figura 2.1: Primera fase

- Segunda fase: Se decidirá el método definitivo a utilizar en la aplicación tras contrastar los diferentes métodos.

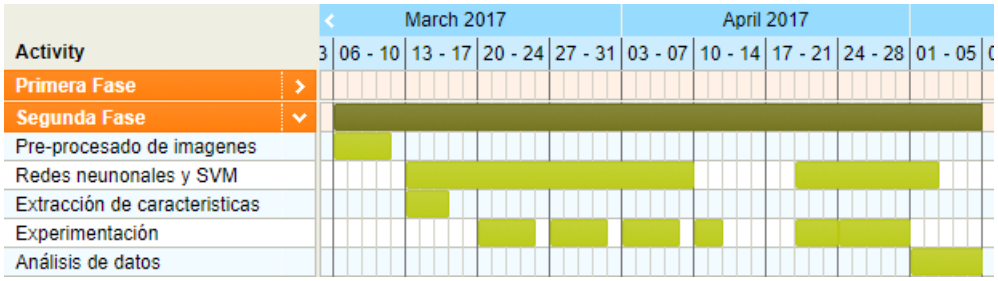


Figura 2.2: Segunda fase

- Tercera fase: Se lleva a cabo el desarrollo de la aplicación y se comprueba su funcionamiento.

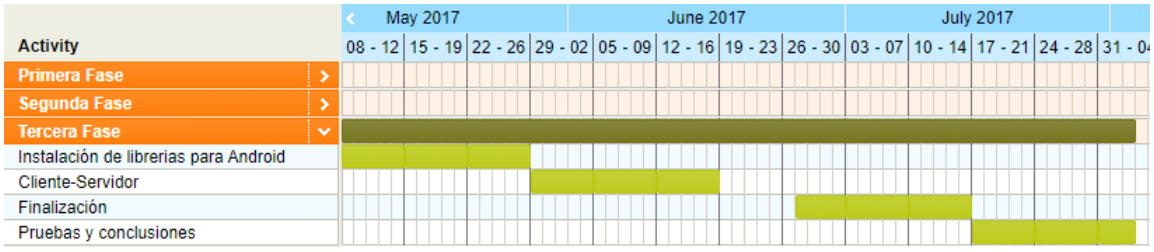


Figura 2.3: Tercera fase

- Cuarta fase: Se termina la memoria y se revisa su contenido.

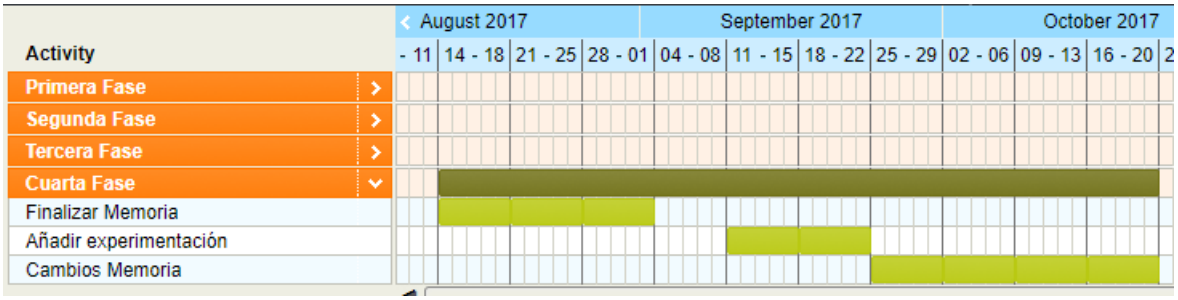


Figura 2.4: Cuarta fase

Para la realización tanto de los gráficos Gantt como para la planificación de las tareas se utilizarán la herramienta Tom's planner [N.V., 2017] y Kanbantool [KT, 2017]. Puesto que ambas son herramientas online y gratuitas.

2.3. Riesgos del proyecto

Como en todo proyecto hay ciertas situaciones de riesgo que pueden llegar a poner en peligro la finalización del mismo en el tiempo acordado. Por lo tanto a continuación se listan los posibles problemas y las medidas que habrá que llevar a cabo con tal de solucionarlos.

- **Estimación errónea:** Como principal riesgo, dada la falta de experiencia en la realización de un proyecto de esta envergadura es posible que la estimación de cada una de las tareas no sea acertada y dure más de lo debido. Para poder cumplir con los plazos aunque haya retrasos se planea terminar el proyecto un par de semanas antes de su entrega para poder suplirlos en caso de necesitarlo.
- **Falta de documentación:** Teniendo en cuenta que la comunicación entre MatLab y Android no se suele llevar a cabo, es posible que nos encontremos con falta de documentación con lo que se dificulte el desarrollo de la aplicación hasta el punto de que algunas de las extensiones sean inútiles. Por ello a pesar de investigar varios métodos para el reconocimiento facial solo se aplicarán aquellos que puedan exportar sus resultados como para que Android sea capaz de reconocerlo.
- **Pérdida de información:** Como en cualquier otro proyecto es posible perder algún tipo de progreso tanto de redacción como de desarrollo debido a cualquier problema o después de realizar cambios decidir que estos no son positivos y querer regresar a un estado anterior. Por ello se realizarán copias incrementales y diferenciales o bien

semanalmente o cada vez que se realice un avance significativo. Estas se almacenarán en la nube, en el ordenador más utilizado y en una memoria USB que no servirá para transportar información sino únicamente para almacenarla y guardarlo en un único lugar.

Además de estos riesgos hay otros como de disponibilidad tanto del alumno como del tutor pero no se han considerado medidas para solucionarlos ya que no ocasionarían problemas demasiado graves como para impedir el avance en el proyecto.

3. CAPÍTULO

Antecedentes

Para la compresión del funcionamiento de las herramientas que se han utilizado durante este proyecto, a lo largo de este capítulo, se ha documentado una de las arquitecturas más importantes del deep learning como son las redes neuronales y las características y desarrollo matemático de las máquinas de vectores de soporte.

3.1. Redes Neuronales

Las redes neuronales biológicas están formadas por neuronas interconectadas entre sí. Cada una de estas neuronas, recibe señales de otras neuronas mediante conexiones sinápticas que pueden ser excitantes o inhibidoras. Dependiendo de las entradas que perciben envían la respuesta a otras neuronas por medio del axón.

Además, contienen un potencial eléctrico interno llamado potencial de membrana. En el momento en el que se supera un umbral, la neurona emite la señal.

Una red neuronal es un conjunto de nodos interconectados de manera similar a las redes neuronales biológicas y su primer modelo matemático lo expusieron Warren McCulloch y Walter Pitts en 1943. Este tipo de redes no son programadas manualmente sino que a partir de una serie de datos se autodefinen. Están organizadas por capas y en cada una de estas, hay un conjunto de nodos que conectan con nodos de otras capas mediante arcos orientados.

Cada uno de estos nodos se asemeja a las neuronas biológicas porque cada una está forma-

da por unos datos de entrada o señales, una función de activación o potencial de membrana y una respuesta o señal de salida. De la misma forma que las conexiones sinápticas pueden modificar la información que transmiten, los datos de entrada se modifican mediante los pesos que se asignan a cada una de las entradas. [Anzola, 2015]

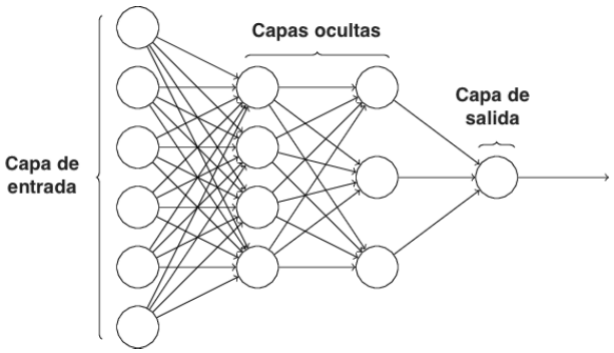


Figura 3.1: Estructura básica

Las capas se clasifican en tres tipos: Entrada, salida y oculta. Todas las redes están formadas por una única capa de entrada y otra de salida y pueden estar compuestas por una o más neuronas. A diferencia de las anteriores, la cantidad de capas ocultas puede variar. Las neuronas de las capas ocultas se comunican entre sí de forma unidireccional, cíclica o recursiva dependiendo del tipo de red neuronal. Así, todos los caminos del grafo parten desde la capa de entrada hasta la capa de salida pasando por las ocultas.

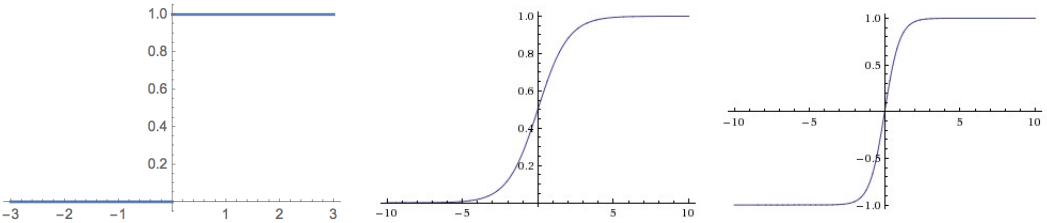
Cada una de los nodos de forma independiente tienen que calcular cuál será su respuesta. Para ello, a cada una de las entradas x_i del nodo hay asociado un peso w_i y cada nodo tiene un sesgo o bias b. Una vez obtienen los datos de entrada, tienen que calcular el producto escalar de estos con sus pesos asociados y sumarle el sesgo.

$$\sum_i w_i \cdot x_i + b \tag{3.1}$$

Después, este resultado se le pasa a la función de activación $\varphi()$ que determinará la respuesta del nodo. Por ello es uno de los principales componentes de una red y dependiendo de cuál se trate obtendrá diferentes resultados ante un mismo entrenamiento. Generalmente su rango de respuesta está comprendido entre [0,1] o [-1,1].

$$\varphi(\sum_i w_i \cdot x_i + b) \tag{3.2}$$

Las funciones de activación más comunes son: La función escalón 3.2a, la función sigmoidea 3.2b y la función tangente hiperbólica 3.2c.



(a) Escalón **(b)** Sigmoidea **(c)** Tangente hiperbólica

$$f(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x < 0 \end{cases} \qquad f(x) = \frac{1}{1/1 + e^{-x}} \qquad f(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{3.3}$$

Figura 3.2: Funciones de activación

Debido a sus características las redes neuronales ofrecen múltiples ventajas. Entre estas están:

- Aprendizaje Adaptativo: Tienen la capacidad de adaptarse o ajustarse por medio de la información que reciben para mejorar su rendimiento.
- Tolerancia a fallos: Son capaces de mantener su rendimiento a pesar de que algunos enlaces o neuronas sean dañados.
- Auto-organización: Son capaces de crear su propia representación de la información durante el aprendizaje.

Las redes neuronales se pueden aplicar a una gran cantidad de ámbitos como pueden ser la modelación económica, las aplicaciones médicas, la optimización de procesos industriales, la seguridad o la investigación científica.

Un ejemplo relacionado con el objetivo de este proyecto sería la visión artificial. Permite reconocer figuras, patrones y objetos en una imagen en base a su apariencia visual. Una de los primeros ejemplos de esto es el reconocimiento de dígitos o letras escritos a mano. Puesto que cada individuo puede representar el mismo dígito con ligeras variaciones, se trata de mediante los patrones que se presentan en estos, poder determinar con precisión de cual se trata.

Las redes neuronales se pueden clasificar según su estructura o topología y el tipo de aprendizaje.

3.1.1. Estructura

- **Aciclicas o Feedforward:** Las conexiones entre capas se realizan a otras que no hayan recibido previamente una señal. Se trata de un tipo de red muy utilizada por su sencillez.

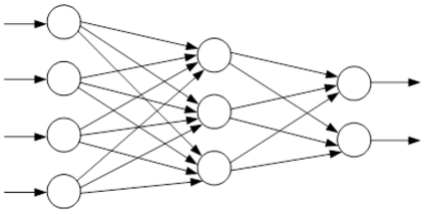


Figura 3.3: Red Feedforward

- **Cíclicas o Feedback:** Las conexiones entre nodos no tienen limitaciones, por lo que un nodo además de con la siguiente capa, puede comunicarse tanto con otro nodo de la misma capa, como consigo mismo o con un nodo de la capa anterior. Debido a esto es necesario añadir la variable del tiempo, puesto que puede haber diferentes estados en un mismo nodo. Gracias a esto aumenta la capacidad de representación de la red pero también se dificulta el proceso de aprendizaje.

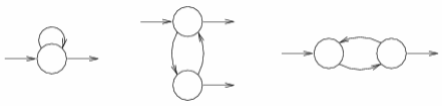


Figura 3.4: Conexiones cíclicas

3.1.2. Tipos de aprendizaje

De entre las propiedades que tiene las redes neuronales, una de las más importantes es la capacidad de aprendizaje. Se basa en adaptar los pesos de la red a partir de estímulos externos. Realizando este proceso de forma iterativa se mejora la respuesta de la red ante nuevas entradas.

Dependiendo del tipo de estímulos externos que recibe la red se pueden distinguir tres aprendizajes.

- **Supervisado:** La red es entrenada por un agente externo que determina cual debería ser la respuesta a una entrada concreta. Los pesos de las neuronas se irán modificando hasta que el error obtenido entre la respuesta objetivo y la generada por la red sea aceptable.
- **No supervisado:** En este caso no hay agente que influencie en el cambio de los pesos. Únicamente se facilitan los datos de entrada y la red dependiendo de la estructura y el algoritmo de aprendizaje obtendrá diferentes salidas. Estas pueden representar por ejemplo grados de similitud entre diferentes entradas o codificaciones simplificadas de las entradas.
- **Reforzado:** Se trata de un aprendizaje similar al supervisado basado en el proceso de prueba y error. Esto es, el agente externo únicamente indica si la respuesta de la red es la esperada. A partir de esto, la red decide en qué medida modificar los pesos.

Entre los algoritmos de aprendizaje supervisados está el de retropropagación o backpropagation que es uno de los más utilizados. Este algoritmo está dividido en dos fases, una hacia adelante y otra hacia atrás. En la primera, se introducen una serie de datos de entrada y se recogen los resultados.

Después, los resultados previos se comparan con los resultados objetivo y se obtiene el error. En la segunda el error se propaga de la salida de la red hacia la entrada modificando los pesos de los nodos de cada capa. Así, al reajustar los pesos se consigue aproximar los resultados futuros al resultado objetivo. Ambas fases se repiten hasta llegar a la condición de parada bien sea por reducir el error o por llegar al número máximo de iteraciones.

3.1.3. Perceptrón

El perceptrón es una red neuronal artificial o la neurona artificial básica que fue desarrollada por Frank Rosenblatt en 1957, en base al trabajo previo de McCulloch y Pitts. Se trata de un clasificador lineal con más de una entrada, con una única salida binaria y sin ninguna capa oculta. A cada entrada x_i está asociado un peso w_i y para que la salida sea positiva o 1 el producto escalar de las entradas y los pesos debe superar el umbral de activación u del perceptrón. [S. Theodoridis, 2009]

$$f(x) = \begin{cases} 1 & \text{si } w \cdot x - u > 0 \\ 0 & \text{c.c.} \end{cases} \tag{3.4}$$

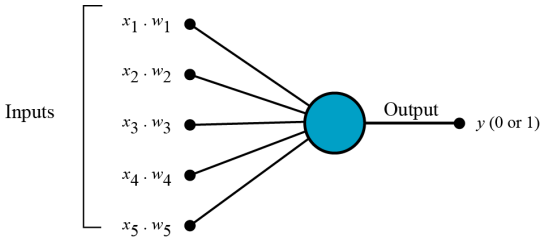


Figura 3.5: Perceptrón

Al contar únicamente con dos capas su capacidad de clasificación es bastante limitada y solo puede representar problemas sencillos como funciones booleanas. Eso sí, no todas las funciones booleanas son representables mediante un perceptrón. Únicamente puede representar funciones linealmente separables debido a que el hiperplano que se obtiene mediante el productor escalar, marca la división entre las 2 clases. Por ello no puede haber representaciones de la misma clase a ambos lados del hiperplano como ocurre con la función XOR.

Por ello, los perceptrones o perceptrones simples están clasificados como clasificadores lineales. Más tarde, estos evolucionaron a clasificadores profundos con lo que incrementaron las posibilidades de las redes neuronales obteniendo clasificadores más precisos.

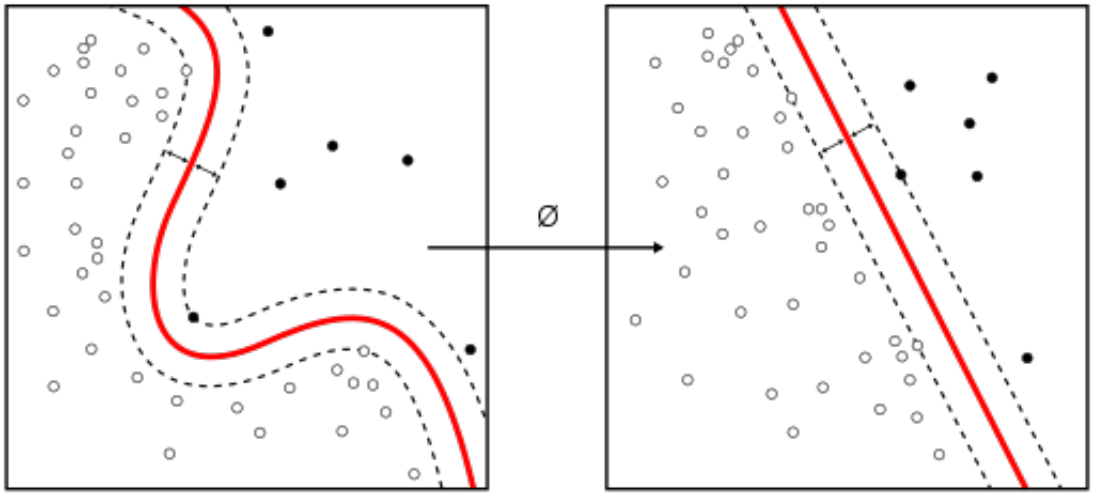


Figura 3.6: Clasificador profundo vs clasificador lineal

Para solucionar esto y conseguir que asimile funciones más complejas se desarrollaron los perceptrones multicapa o MLP que básicamente son perceptrones normales pero con capas ocultas y diferentes funciones de activación. Estos son capaces de delimitar regiones no linealmente separables puesto que generan múltiples hiperplanos. Tal y como se muestra en la figura 3.7 según se aumenta el número de capas las regiones son más definidas.

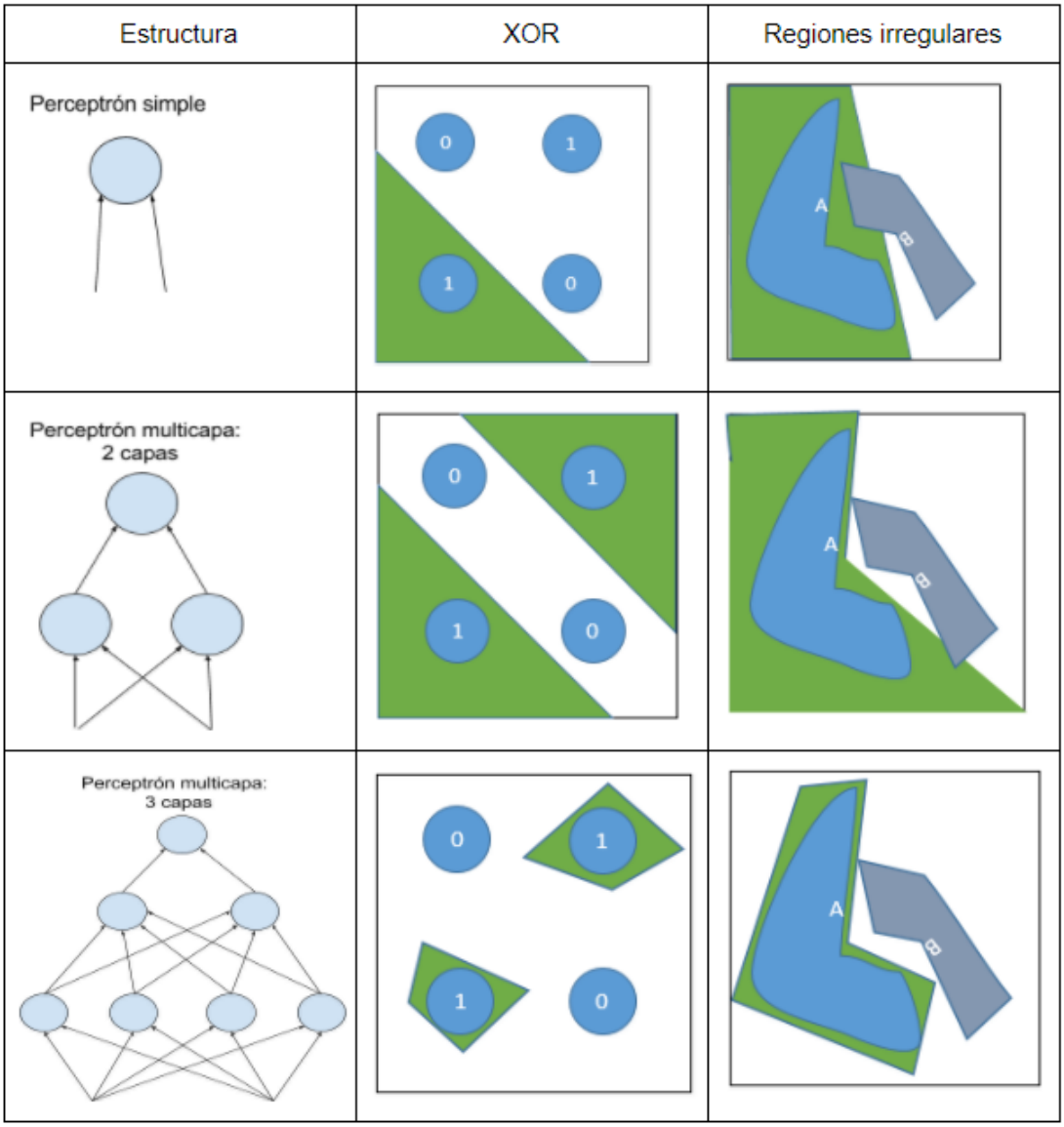


Figura 3.7: Regiones de decisión

3.1.4. Red Neuronal Convolucional

Se trata de una red Feedforward especializada en el análisis de imágenes. Por ello la red utilizada en este proyecto es de este tipo. Dado que está pensada para el procesamiento de imágenes su estructura cambia respecto a una red convencional y en vez de introducir un vector a la red, se introduce la matriz de la imagen. Esto ocurre porque mediante filtros y submuestreos se va reduciendo el tamaño de la imagen hasta lograr una representación mínima con la menor perdida de información posible. [Pokharna, 2016]

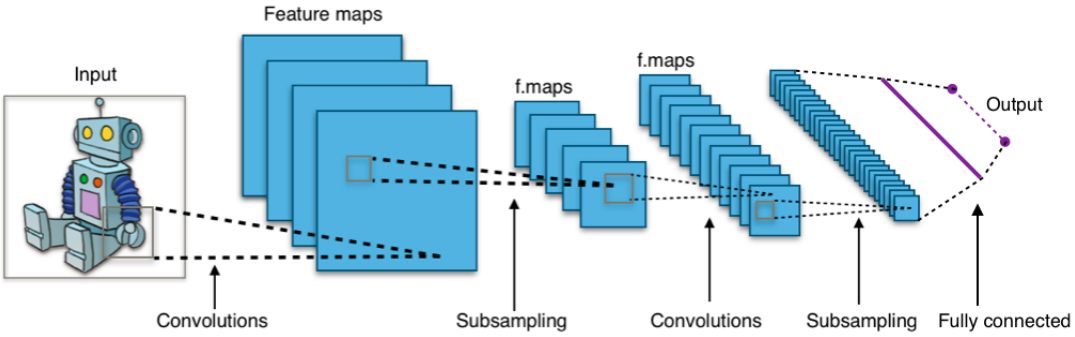


Figura 3.8: Red neuronal convolucional

Además, la estructura interna también cambia puesto que hay diferentes tipos de capas ocultas. Dado que hay diferentes capas, el diseño de estas redes es más complejo y de esto depende la eficiencia.

3.1.4.1. Capas ocultas

Existen 3 tipos de capas ocultas y cada una de ellas tiene un funcionamiento diferente:

- **Convolucional:** Recibe como entrada una imagen, a la cual, se le aplica un filtro para obtener un representación más pequeña denominada mapa de características. Gracias a esto las redes convolucionales pueden procesar imágenes de gran tamaño de forma más sencilla.
  Además, la función de activación de estas capas suele ser *rectified linear unit* (ReLu) 3.9 que incrementa las propiedades no-lineales de la función de decisión y de toda la red.
- **Reducción o Pooling:** Se suele situar después de la capa convolucional y sirve para reducir el tamaño del mapa de características. Para ello, realizan un submuestreo

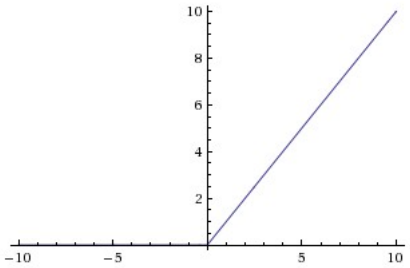


Figura 3.9: Función de activación: $f(x) = \max(0, x)$

y así reducen el cálculo de las siguientes capas y ayudan a reducir el overfitting o sobreajuste.

- **Totalmente conectada:** Estas capas se encuentran después de todas las convolucionales y de reducción. Llegados a este punto los pixeles se consideran como nodos independientes como en una red neuronal corriente.

3.2. Support Vector Machine (SVM)

3.2.1. Introducción

Las máquinas de vectores de soporte o SVM son un clasificador lineal basado en margen máximo a partir de dos vectores de soporte. La solución de las SVM será un hiperplano que divida en 2 el espacio. Para obtenerlo se tienen en cuenta unas muestras con unas características específicas que forman los vectores de soporte. Estas muestras son las que se encuentran en el extremo del conjunto que más cerca está del otro. El espacio entre los vectores de soporte de cada clase se conoce como margen máximo y está vacío de muestras. Este margen se obtiene a partir de la solución de un problema de optimización cuya función a optimizar es precisamente el margen. [Valveny, a] [Valveny, b]

Debido a estas características las SVM tienen una cierta robustez estadística, sobre todo al sobreajuste puesto que su solución se modifica únicamente cuando los vectores de soporte cambian y como están constituidos por una pequeña cantidad de muestras solo cuando estas cambien o aparezca una nueva muestra en el margen se modificará este.

El principal inconveniente de las SVM tal y como se han explicado hasta ahora es que únicamente son válidos para conjuntos linealmente separables y también completamente separables sin solapamiento entre clases. Para solucionar esto hay dos opciones:

- **Relajar la condición del margen:** Añadir una tolerancia a errores a las SVM de forma que algunas muestras queden erróneamente clasificadas con tal de separarlo de forma lineal.

- **Transformar el espacio de características:** Algunos conjuntos de datos que a priori no son linealmente separables se pueden volver separables en un espacio de dimensión superior. Para ello hay que utilizar las funciones denominadas kernel.

3.2.1.1. SVM multiclase

Además de para generar clasificadores binarios las SVM pueden utilizarse para realizar clasificaciones multiclase. Para ello es necesario utilizar más de un clasificador que en conjunto realizan la función deseada. Básicamente hay dos formas de representar estos clasificadores:

- **One vs One:** Se trata de entrenar un clasificador por cada combinación posible de dos clases. En este caso habrá que generar $n * \frac{(n-1)}{2}$ clasificadores. A la hora de clasificar las nuevas muestras hay que clasificar la muestra en todos los clasificadores y la clase en la que más veces haya sido clasificada será la clase resultante.

- **One vs The rest:** Este método se basa en generar un clasificador por cada una de las clases contraponiendo como muestras negativas al resto de clases. Por ello se generaran n clasificadores. Una vez generados para determinar la clase de una nueva muestra se aplican todos los clasificadores y se obtienen los valores normalizados de cada clasificador y el clasificador cuyo valor normalizado sea el mayor será la clase a la que pertenece la muestra.

3.2.2. Desarrollo matemático

Dado que se trata de un clasificador lineal la formulación matemática de la solución será la siguiente:

$$\mathbf{w}^T \mathbf{x}_i + b = 0 \tag{3.5}$$

$\mathbf{w}$ es el vector ortogonal al hiperplano y b es su posición en el espacio. Teniendo en cuenta que las muestras positivas se obtendrán cuando obtengamos un valor superior a 0 y las negativas cuando obtengamos un valor inferior a 0 la función de clasificación será la

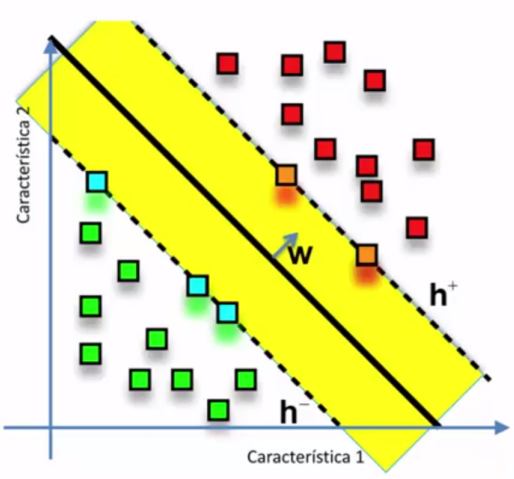


Figura 3.10: SVM

siguiente:

$$f(\mathbf{x}) = sgn(\mathbf{w}^T \mathbf{x}_i + b) \tag{3.6}$$

Esta determinará la clase a la que pertenece una muestra respecto a su signo. El hiperplano solución es el hiperplano medio entre los dos hiperplanos que contienen los vectores de soporte. El margen podrá calcularse a partir de la distancia entre los vectores de soporte. Al ser paralelos al hiperplano solución se pueden representar como h^+ y h^- y su distancia al hiperplano será la siguiente:

$$\begin{aligned} h^+ &= \mathbf{w}^T \mathbf{x}_i + b = +1 \\ h^- &= \mathbf{w}^T \mathbf{x}_i + b = -1 \end{aligned} \tag{3.7}$$

$$d^+ = d^- = \frac{|\mathbf{wx} + b|}{\| \mathbf{w} \|} = \frac{1}{\| \mathbf{w} \|} \tag{3.8}$$

Teniendo en cuenta la distancia de los vectores al centro se puede obtener la función que define el margen y así la función a optimizar.

$$margen = d^+ + d^- = 2\frac{1}{\| \mathbf{w} \|} \tag{3.9}$$

Dado que todas las muestras tienen que encontrarse más allá del margen todas ellas tienen que estar sujetas a la siguiente ecuación.

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \tag{3.10}$$

Para obtener la solución se puede maximizar dicha función o lo que es lo mismo minimizar el problema inverso. Esta minimización tiene que estar sujeta a la misma condición 3.10 que la función a maximizar, con lo que se trata de realizar una optimización cuadrática sujeta a una inecuación.

$$\phi(\mathbf{w}) = \frac{1}{2} \parallel \mathbf{w} \parallel^2 \tag{3.11}$$

Para ello utilizaremos una función auxiliar conocida como Lagrangiano. Está construido a partir de la suma de la función a optimizar y de las restricciones del problema multiplicados por los coeficientes α o dicho de otra forma los multiplicadores de Lagrange.

$$L(x, \alpha) = f(x) + \sum_i \alpha_i g_i(x) \qquad \forall \alpha_i \geq 0 \tag{3.12}$$

Tras realizar esta transformación la solución se va a obtener minimizando el Lagrangiano respecto a las variables $\mathbf{w}$ y b y maximizarlo respecto a α. Una vez sustituidas la función a optimizar y la restricción obtenemos la siguiente expresión:

$$L(x, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_i \alpha_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1] \tag{3.13}$$

La minimización del Lagrangiano implica que las derivadas parciales respecto a las variables $\mathbf{w}$ y b sean iguales a 0.

$$\begin{aligned} \mathbf{w} &= \sum_i \alpha_i y_i \mathbf{x}_i \\ \sum_i \alpha_i y_i &= 0 \end{aligned} \tag{3.14}$$

Una vez obtenido estas derivadas parciales sustituyendo en la expresión anterior 3.13 obtenemos la función a maximizar y a qué está sujeta. Esta expresión se denomina support vector machine dual que proviene del problema primal. Su resolución se puede llevar a cabo con algún método numérico estándar.

$$L(x,b,\alpha) = -\frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_i \alpha_i$$
$$Sujeto\ a \sum_i \alpha_i y_i = 0 \tag{3.15}$$

Una vez resuelto finalmente se obtienen los multiplicadores de Lagrange con los cuales se procede a obtener el vector ortogonal **w** y el coeficiente b con los cuales ya se podrán clasificar las futuras muestras.

3.2.2.1. Relajación del margen

Para relajar la condición del margen hay que añadir variables de holgura que permitirán que algunas muestras violen la condición del margen. Estas se añaden a las condiciones que tienen que cumplir las muestras de entrenamiento. Esto implica que haya que añadir un parámetro de regularización C para así controlar la maximización del margen aplicando el mínimo error posible. Los valores de este estarán dentro del rango $[0,\infty)$ y el clasificador obtenido utilizando este se denominará C-SVM. El valor de este parámetro se calculará de manera estándar a partir de validación cruzada.

Para simplificar la obtención del parámetro de regularización Schölkopf et al. reformulo las SVM añadiendo el parámetro ν. En este caso está definido en el rango [0,1] y su clasificador se denomina nu-SVM. En las siguientes explicaciones se dejará de lado esta formulación centrándose en la original.

Figura 3.11: Relajación del margen

Al llegar a la representación del problema dual, a las condiciones que este tiene que cumplir, se añade una nueva inecuación que regula la contribución de los vectores que tienen α diferentes de 0. Una vez obtenidas los multiplicadores de Lagrange la solución final del clasificador no incluye las variables de holgura ni el parámetro C siendo tal y como era en la anterior representación 3.6.

$$Variables\ de\ holgura: \ \xi_i \geq 0 \tag{3.16}$$

$$\begin{gathered} SVM\ Primal: \\ Minimizar\ \Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i \\ Sujeto\ a: \ y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \end{gathered} \tag{3.17}$$

$$\begin{gathered} SVM\ Dual: \\ Maximizar\ \Theta(\alpha) = -\frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j + \sum_i \alpha_i \\ Sujeto\ a: \ \sum_i \alpha_i y_i = 0 \quad y \quad 0 \leq \alpha_i \leq C \end{gathered} \tag{3.18}$$

3.2.2.2. Funciones Kernel

La forma de aumentar la dimensionalidad a las SVM sería utilizando una función de mapeo ρ. Pero dado que en el problema dual los vectores solo aparecen como productos escalares, se trata de utilizar una función Kernel que represente el producto escalar en el espacio mapeado por ρ. Gracias a esto no es necesario definir explícitamente la función ρ con lo que será suficiente con encontrar la función Kernel entre dos muestras. Ya que representa un producto escalar la sustitución en la función a optimizar y en la función a clasificar será directa.

$$
\begin{aligned}
&SVM\ Dual: \\
&Maximizar\ \Theta(\alpha) = -\frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j) + \sum_i \alpha_i \\
&f(\mathbf{x}) = sgn(\sum_i \alpha_i y_i K(\mathbf{x},\mathbf{x}_i) + b)
\end{aligned}
\tag{3.19}
$$

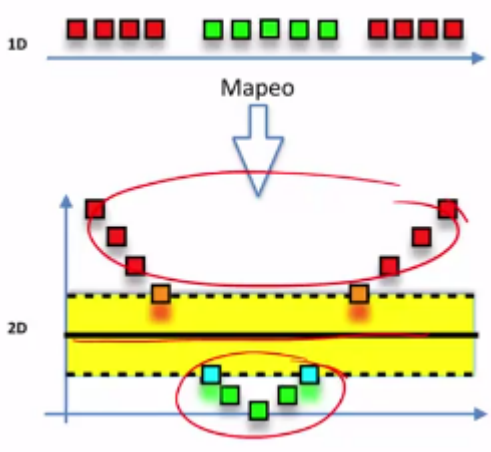


Figura 3.12: Kernel

Hay diferentes tipos de funciones Kernel lo que facilita la selección de la función Kernel adecuada para cada problema.

4. CAPÍTULO

Experimentación

El objetivo principal de este proyecto es lograr diferenciar a un individuo del resto por medio de una imagen facial. A ese procedimiento se le llama supervisión y consiste en comparar las características de la imagen, con las de imágenes previamente registradas a las que se les ha indicado a que clase pertenecen. Tras compararla se define su clase dependiendo de a qué conjunto se asemeje más. Para ello, se utilizan en conjunto las redes neuronales y las SVM. En este proceso se distinguen dos fases: Entrenamiento y supervisión.

En la primera fase gracias a la red VGG-Face [Parkhi et al., 2015] se obtienen los vectores de características de diferentes individuos y del individuo a verificar. Estos vectores al haberse obtenido por medio de esa red serán de 4096 características y cada uno representará de forma única cada uno de los rostros. Con todos ellos se entrenaran las diferentes SVM, indicando a que clase pertenece cada uno de los vectores, que tipo de SVM utilizar y con qué tipo de Kernel.

En la fase de supervisión, a las SVM se les facilitará un vector de características y determinará por medio de una respuesta binaria si el individuo corresponde con el del entrenamiento. La funcionalidad de las SVM se hará con la herramienta LibSVM [Chang and Lin, 2000]. Su funcionamiento se detalla en el apéndice B. Para determinar que configuración de la herramienta obtiene mejores resultados se llevarán a cabo una serie de pruebas

Las configuraciones estarán compuestas de 2 tipos de SVM de clasificación combinándolas con los diferentes tipos de kernel que trae por defecto la herramienta. Para ello se realizarán 4 pruebas con cada configuración. Estas irán reduciendo la cantidad de mues-

tras necesarias para obtener el clasificador. La primera de ellas constará de 50 vectores de un individuo y 100 de diferentes individuos. En las siguiente dos pruebas se dividirán los conjuntos por la mitad sucesivamente. La última prueba constara de un total de 25 vectores de los cuales únicamente 4 serán del usuario. Esto será así para que la aplicación final pueda funcionar con la mínima cantidad de imágenes posible.

Tras generar todos los clasificadores, se testearán con un conjunto de 50 vectores diferentes a los de entrenamiento, de los cuales la mitad serán del mismo individuo que en la clasificación y las demás de otros.

En las tablas 4.1 y 4.2 se muestran los resultados de las pruebas realizadas con los 2 clasificadores y todos los tipos de kernel, indicando el porcentaje de acierto o precisión (%) y el error cuadrático medio.

	Prueba 1		**Prueba 2**		**Prueba 3**		**Prueba 4**	
	Precisión	**Error**	**Precisión**	**Error**	**Precisión**	**Error**	**Precisión**	**Error**
Lineal	100	0	100	0	100	0	98	2,42
Polinomial	100	0	100	0	96	4,84	94	7,26
Base radial	98	2,42	91,67	10,08	90	12,1	84	19,36
Sigmoide	100	0	100	0	100	0	100	0

Tabla 4.1: C-SVM con todos los tipos de Kernel.

	Prueba 1		**Prueba 2**		**Prueba 3**		**Prueba 4**	
	Precisión	**Error**	**Precisión**	**Error**	**Precisión**	**Error**	**Precisión**	**Error**
Lineal	98	2,42	98	2,42	98	2,42	98	2,42
Polinomial	92	9,68	94	7,26	96	4,84	70	36,3
Base radial	90	12,1	88	14,52	86	16,94	84	19,36
Sigmoide	100	0	100	0	100	0	100	0

Tabla 4.2: nu-SVM con todos los tipos de Kernel.

Además de realizar estas pruebas, con el fin de encontrar alternativas al uso de redes convolucionales para extraer las características de la imagen, se realizarán otras pruebas. Estas estarán compuestas por los histogramas convencionales y los histogramas de gradientes orientados (HOG)[Mallick, 2016].

- Para obtener el histograma convencional primero se escalará la imagen de la misma manera que se realizó con las redes convolucionales y se obtendrá la imagen en escala de grises. Una vez hecho esto mediante la función *imhist* de Matlab se

obtendrá el histograma de la imagen que se representa en 256 intervalos, uno por cada nivel de gris.

- El histograma de gradientes orientados, que a diferencia del anterior que enumera las apariciones de cada nivel de gris, representa la imagen mediante las diferencias de intensidad entre los pixeles de alrededor. Para ello, primero la imagen se divide en fragmentos o celdas y en cada una de ellas se calcula el histograma de gradientes de forma local, que en el caso de matlab por defecto constara de 9 intervalos. Estos intervalos agrupan los diferentes ángulos que pueden tener los gradientes. Así, cada pixel dependiendo de la dirección del gradiente añadirá la magnitud de este al intervalo que le corresponda del histograma.

  Una vez obtenido el histograma de todas las celdas se agrupan en grupos de 4 creando bloques. Cada uno de estos bloques está compuesto por 4 histogramas de 9 posiciones o lo que es lo mismo un vector de 36 posiciones. El vector de cada bloque se normaliza y finalmente se agrupan en un único vector obteniendo el HOG final.

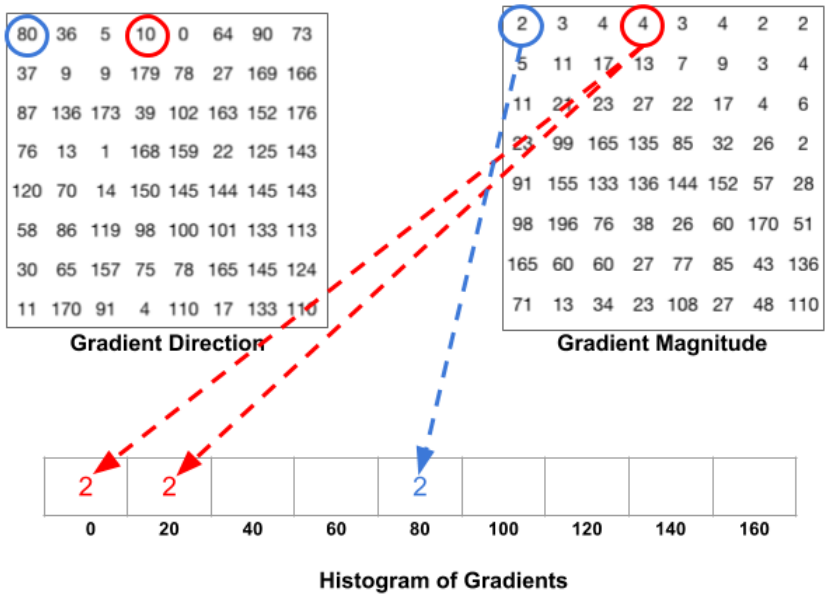


Figura 4.1: HOG

Con los histogramas se realizará la última de las pruebas realizadas con las SVM. No se realizarán pruebas adicionales puesto que la utilización de las redes convolucionales de cara a la aplicación final, es uno de los objetivos de este proyecto. Aun así se realizan puesto que es conveniente comparar sus resultados.

En las tablas 4.3 y 4.4 se muestran los resultados de las pruebas adicionales indicando el porcentaje de acierto o precisión (%) y el error cuadrático medio de cada uno de los métodos.

	CNN		**Histograma**		**HOG**	
	Precisión	**Error**	**Precisión**	**Error**	**Precisión**	**Error**
Lineal	98	2,42	94	7,26	100	0
Polinomial	94	7,26	86	16,94	50	60,5
Base radial	84	19,36	50	60,5	50	60,5
Sigmoide	100	0	50	60,5	50	60,5

Tabla 4.3: C-SVM: CNN vs histograma vs HOG.

	CNN		**Histograma**		**HOG**	
	Precisión	**Error**	**Precisión**	**Error**	**Precisión**	**Error**
Lineal	98	2,42	90	12,1	100	0
Polinomial	70	36,3	94	7,26	100	0
Base radial	84	19,36	50	60,5	100	0
Sigmoide	100	0	50	60,5	100	0

Tabla 4.4: nu-SVM: CNN vs histograma vs HOG.

5. CAPÍTULO

Aplicación

El objetivo principal era desarrollar una aplicación capaz de verificar al usuario en comparación con el propietario, mediante una fotografía. Para ello, se iba a tratar de integrar en el terminal una red convolucional similar a VGG-Face [Parkhi et al., 2015] pero de menor tamaño. Esta red se encargaría de extraer las características de la fotografía. Después, por medio de un clasificador se determinaría si las características eran las mismas que las del propietario.

Inicialmente, los datos de entrada del clasificador se iban a obtener en Matlab mediante VGG-Face y MatConvNet. Para después transferirlos al dispositivo y allí generar el clasificador. Según fue avanzando el proyecto, se llegó a la conclusión de que si la base del clasificador se obtenía a través de VGG-Face era necesario utilizarla también para poder realizar la verificación. Por ello, se decidió cambiar el funcionamiento.

Finalmente, se decidió que en Matlab se procesarían las imágenes con el rostro ya preprocesado y que lo único que se haría sería obtener los vectores de características y guardarlos en ficheros. Todas las demás operaciones se realizarían en el dispositivo.

Se han utilizado tres entornos base. Estos son Android Studio, Matlab y Wampp server. A su vez tanto en Android Studio como en Matlab se han tenido que añadir complementos adicionales. Por un lado, en Android Studio se ha añadido la librería NDK que permite ejecutar código C++ desde Android. Gracias a esta, se puede añadir la librería LibSVM [Tung, 2015] con el fin de generar las SVM y verificar al individuo en el propio dispositivo. Por otro lado en Matlab se ha añadido MatConvNet para la obtención de los vectores de características y LibSVM [Chang and Lin, 2000] para la realización de las pruebas

iniciales. El procedimiento seguido para su adición esta detallado en el Anexo A y B.

Las imágenes tanto para realizar las pruebas como para generar el clasificador final provienen de la base de datos de MORPH [MOR, 2017] [DBW, 2008] y fueron pre-procesadas por Gotzal Ruiz.

5.1. Casos de uso

Aunque al principio se pretendía añadir otros casos de uso, tras realizar pruebas se concluyó que su desarrollo necesitaba de un alto conocimiento en Android puesto que dependían de funcionalidades poco convencionales. Por ello, se decidió realizar 3 casos de uso.

- **Generar las SVM**: El usuario realiza una serie de fotografías hasta que la aplicación obtiene las suficientes y las envía al servidor. Este envía de vuelta un archivo con todos los vectores de características tanto los de las fotografías como los de un subconjunto de la base de datos. Por ultimo genera las SVM y después este caso de uso se bloquea y se desbloquean los demás. C.1
- **Eliminar las SVM**: Elimina todo el contenido relacionado con el usuario registrado. Tras eliminarlos tanto este caso de uso como el de verificación se bloquean y se desbloquea el que genera las SVM.
- **Verificar al usuario**: El usuario realiza una fotografía y esta se envía al servidor para obtener a cambio un archivo con el vector de características. Este se verifica mediante las SVM generado previamente e indica si se trata del mismo usuario. C.2

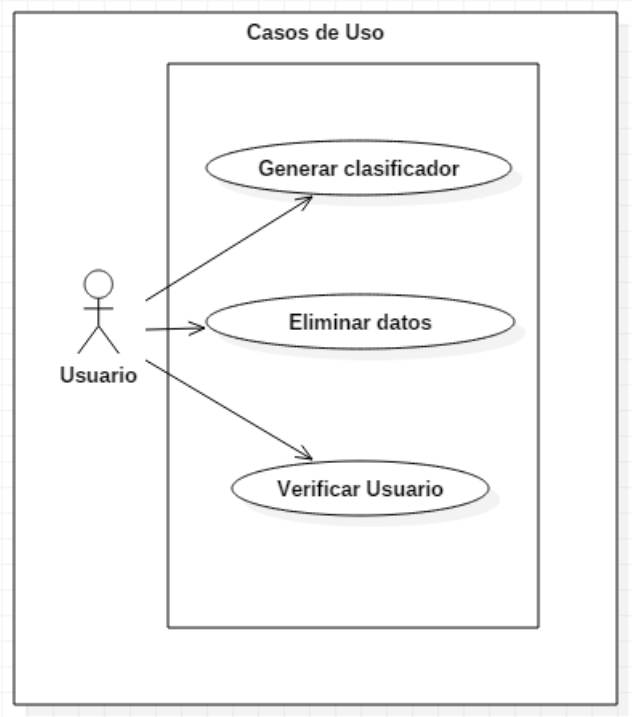


Figura 5.1: Casos de uso

5.2. Sprints

El desarrollo de la aplicación se ha llevado a cabo en tres diferentes sprints comprendidos en las primeras tres fases del proyecto. Estos son: La implementación de las librerías, la comunicación cliente servidor y la puesta en común de todos los apartados en conjunto con la realización de pruebas. Aunque en un principio no se consideró a las primeras dos fases parte del desarrollo, para la comprensión de este se han añadido al primer sprint.

5.2.1. Librerías

Este primer sprint a su vez quedo dividida en dos, una por cada herramienta. Primero se llevó a cabo todo lo pertinente a Matlab ya que las pruebas iniciales se realizarían en este programa. Una vez añadida MatConvNet, descargada la red VGG-Face y tras leer la documentación se programó una función para recopilar los vectores de características en una matriz que serviría para la posterior creación de las SVM.

Durante este tiempo se investigó acerca del funcionamiento de las SVM y las redes neu-

ronales. Para así poder comprender el funcionamiento de la aplicación final.

Tras añadir LibSVM se modificó la función de extracción de características para poder pasarle los datos de forma adecuada tanto a la función de entrenamiento, como a la función de predicción. Se tuvo que generar además de la matriz de vectores un vector adicional en el que se indicará la clase de cada individuo.

Una vez hecho esto se realizaron unas pruebas variando la cantidad de individuos tanto positivos como negativos. Tal y como se puede observar en la sección 4, con muy pocos individuos de entrenamiento los resultados de las predicciones eran muy positivos.

Debido a que es necesario conectarse con Matlab para obtener los vectores de características de la red VGG-Face, se trató de crear una red sustituta de menor tamaño para así añadirla al teléfono y no tener que depender de Matlab. Después de realizar diferentes pruebas se llegó a la conclusión de que la red inicial era lo suficientemente compleja y trabajada como para conseguir reducirla sin realizar un estudio profundo de la estructura de la red original.

Por ello finalmente se decidió obtener todos los vectores de características por parte de Matlab.

5.2.2. Cliente-Servidor

Para la realización de este sprint se investigaron diferentes formas de comunicación tanto con una conexión directa con un socket como con una conexión HTTP. Finalmente se decidió utilizar una conexión HTTP gestionada a través de WAMP server con un script en php. Para ello se utilizó un documento que incluía los pasos para realizar una aplicación en Android de procesado de imágenes [Stanford, 2015].

Utilizando dicho documento como base, se añadió a la aplicación en desarrollo las funciones imprescindibles de comunicación y se modificaron las de tratamiento de datos, puesto que la aplicación original enviaba una imagen desde el móvil y esperaba esa imagen modificada de vuelta. La cuestión es que en nuestro caso lo que espera el dispositivo depende del caso de uso. Puede ser un archivo de texto, con los vectores de entrenamiento o un único vector para la verificación.

También se tuvo que añadir una librería nativa de Android para la detección facial. No dispone de demasiadas características pero son suficiente para identificar los rostros en una fotografía y obtener datos barométricos sobre ellos.

Para terminar de realizar la comunicación se tuvieron que realizar una serie de funciones en Matlab para terminar de procesar la imagen subida al servidor y generar un archivo con el vector de características.

5.2.3. Procesado final

Durante este último sprint, se terminaron de cuadrar todas las herramientas para que tanto la comunicación como la generación de archivos fuera correcta.

Primero, se terminaron de ajustar los tiempos de reacción de los botones de la aplicación. Esto fue necesario porque antes de poder procesar los vectores de características en el dispositivo, se tenía que terminar de procesar las imágenes en Matlab para que finalmente la función de php lograra enviar los archivos.

Después, se tuvo que realizar el pre-procesado de la imagen para que llegara a Matlab directamente recortada y así ahorrar la mayor cantidad de ejecuciones posibles en el servidor. Una vez hecho esto, se tuvo que estructurar un método para poder distinguir el tipo de imagen que se estaba subiendo al servidor, puesto que solo se utilizó una única función php.

Se tuvieron en cuenta 4 posibilidades:

- La fotografía realizada no contiene una cara con lo que hay que repetirla.
- Las SVM ya estaba generado con lo que el archivo solo tenía que contener un vector de características.
- Se había eliminado las SVM y había que generar uno nuevo.
- La generación de las SVM estaba en proceso.

Se decidió asignar un nombre diferente a cada una de estas posibilidades para que Matlab pudiera diferenciarlas. Tras solventar esto, la aplicación quedo terminada y se efectuaron una serie de pruebas para comprobar que el rendimiento concordaba con la experimentación. Estas se hicieron con 3 individuos diferentes y los resultados fueron los esperados. El único inconveniente fue que la librería de Android que detecta la cara tenía dificultades en detectar aquellas con mucho bello facial.

6. CAPÍTULO

Problemas

Es inevitable que durante proyectos de larga duración se presenten complicaciones. En este caso el primer problema fue con la versión de la herramienta NDK de Android Studio. La última versión requería más configuraciones que en la versión en la que se implementó libSVM. Por ello se tuvo que investigar la estructura de las dos versiones de NDK para poder añadirlo. A esto se le sumo la falta de experiencia en Android que dificulto su compresión y el desarrollo general de la aplicación.

Durante la experimentación, al realizar el entrenamiento de las nu-SVM de menor tamaño, un error impedía generarlas. Tras revisar el código de las funciones, se comprendió que para utilizar el valor de ν por defecto (0.5), tenía que cumplirse una condición. La media de muestras de cada clase multiplicado por ν tenía que ser inferior al número de muestra del conjunto más pequeño. Por ello en la generación de estas SVM se aplicó un $\nu = 0,3$.

Por último, la estimación de los tiempos de algunas tareas del proyecto no fue la adecuada. A consecuencia de esto la fecha de finalización se tuvo que retrasar. Esto ocurrió debido a que los problemas ya mencionados junto con los intentos de reducir la red VGG-Face y el intentar añadir más funcionalidades duraron más de lo esperado.

7. CAPÍTULO

Conclusiones y lineas futuras

7.1. Conclusiones del proyecto

En términos generales, los resultados de la aplicación son positivos puesto que tiene un alto porcentaje de acierto y los fallos ocurren principalmente debido a que las librerías nativas de Android no reconocen que en la fotografía haya una cara.

Por otro lado, los tiempos de ejecución de las SVM son muy pequeños y tienen un bajo porcentaje de fallo en la clasificación de los vectores provenientes de VGG-Face. Por ello se podría considerar que los objetivos del proyecto han sido completados.

Además, tal y como se muestra en la experimentación, los resultado obtenidos mediante los histogramas de gradientes orientados en conjunto con las SVM son sorprendentes. Puesto que, como se muestra en la tabla 4.4 su precisión es del 100 % con todos los tipos de kernel.

7.2. Conclusiones personales

A partir de un conocimiento superficial sobre aplicaciones Android, según se avanzaba, quedaba claro que Android no es tan sencillo como pudiera parecer, al menos no cuando se trata de conjugar diferentes características del dispositivo para que trabajen de forma conjunta.

Además durante el proyecto se ha asimilado el funcionamiento y las posibilidades que ofrecen tanto las redes neuronales como las máquinas de vectores de soporte. Aunque la funcionalidad de la aplicación se ha conseguido, el trabajo solo ha supuesto una iniciación en las redes neuronales, quedando mucho más que aprender.

Finalmente a pesar de la falta de experiencia en trabajos de esta dimensión, de haber tenido una serie de contratiempos y de haber tenido que alargar la duración del mismo, el resultado ha sido muy satisfactorio. No solo lo ha sido desde el punto de vista de la informática, sino también desde el de la organización, planificación, estudio y realización de un proyecto.

7.3. Lineas futuras

- Reducir los tiempos de ejecución de determinadas funciones.
- Trasladar todos los cálculos al dispositivo generando una red que sustituya a la VGG-Face y que pueda ser almacenada en este.
- Investigar y experimentar a cerca de otros métodos de obtención de vectores de características.
- Añadir casos de uso relacionados con el bloqueo de aplicaciones y desbloqueo mediante la supervisión.
- Implementar un detector de rostros propio para prescindir del de Android.

Anexos

A. ANEXO

Herramientas MatLab

Una vez realizada la instalación base de MatLab R2015a, procedemos a la descarga de la herramienta de MatConvNet de su página oficial. Esta la añadimos a la carpeta donde se encuentran el resto de herramientas y para poder comenzar a utilizarla, primero hay que compilarla, en nuestro caso con el Visual Studio 2013. Después de esto ya podemos comenzar a usarla normalmente.

Para llevar a cabo una prueba, primero tendremos que obtener una red neuronal pre-entrenada en nuestro caso VGG-Face que obtendremos de la propia página de MatConvNet. Tras cargar tanto la red como la imagen en cuestión que queremos procesar, hay que normalizar la imagen para lo que hay que re-dimensionar y ajustar la imagen con la media de cada uno de los componentes del RGB.

Una vez realizado esto, en la variable *res* obtenemos el resultado que se trata de una estructura divida en 38 capas. La capa que se ha utilizado para este proyecto ha sido la 36 que contiene el vector de características de la imagen.

El script necesario para la realización de este proceso es el siguiente:

```
run matlab/vl_compilenn;
run matlab/vl_setupnn;

net = load('vgg-face.mat');
im = imread('imagen-a-estudiar.jpg') ;

im_ = imresize(single(im), net.meta.normalization.imageSize(1:2)) ;

for i = 1:3
```

```
10 im_(:,:,i) = im_(:,:,i) - net.meta.normalization.averageImage(:,:,i) ;
11 end
12
13 res = vl_simplenn(net, im_) ;
```

B. ANEXO

Software Android

Para poder utilizar las máquinas de vectores de soporte en Android es necesario añadir un software específico. Aunque hay varias opciones la que mayor soporte tiene es LibSVM [Chang and Lin, 2000]. Además, como está disponible para diferentes plataformas se añadirá también a Matlab para así poder realizar las pruebas iniciales en ella.

Se puede integrar en una aplicación sin necesidad de instalar aplicaciones adicionales en el dispositivo puesto que su código completo está disponible en GitHub.

Una vez implementado consta de dos funciones fundamentales. Una para entrenar el SVM y otra para predecir y ambas tienen un único parámetro. Es un String en el que se indica el tipo de SVM, la ruta a las muestras clasificadas y la ruta en la que se guardara el modelo para el caso de la función de entrenamiento y la ruta a la nueva muestra a clasificar, la ruta al modelo y la ruta en la que se almacenará el resultado para la función de perdición.

$jniSvmTrain$(”$opciones\ RutaAMuestras\ RutaAlmacenamientoDelModelo$”);

$jniSvmPredict$(”$RutaMuestraAClasificar\ RutaAlModel\ RutaResultado$”);

Entre las opciones a la hora de generar el SVM se encuentran:

- **Tipo de SVM:** Hay 5 posibles tipos de SVM que permite generar LIBSVM. En nuestro caso utilizaremos principalmente las opciones de C-SVM y nu-SVM que implementa la relajación del margen.
- **Tipo de Kernel:** Dispone de 4 Kernels diferentes: Lineal, Polinomial, Fución de

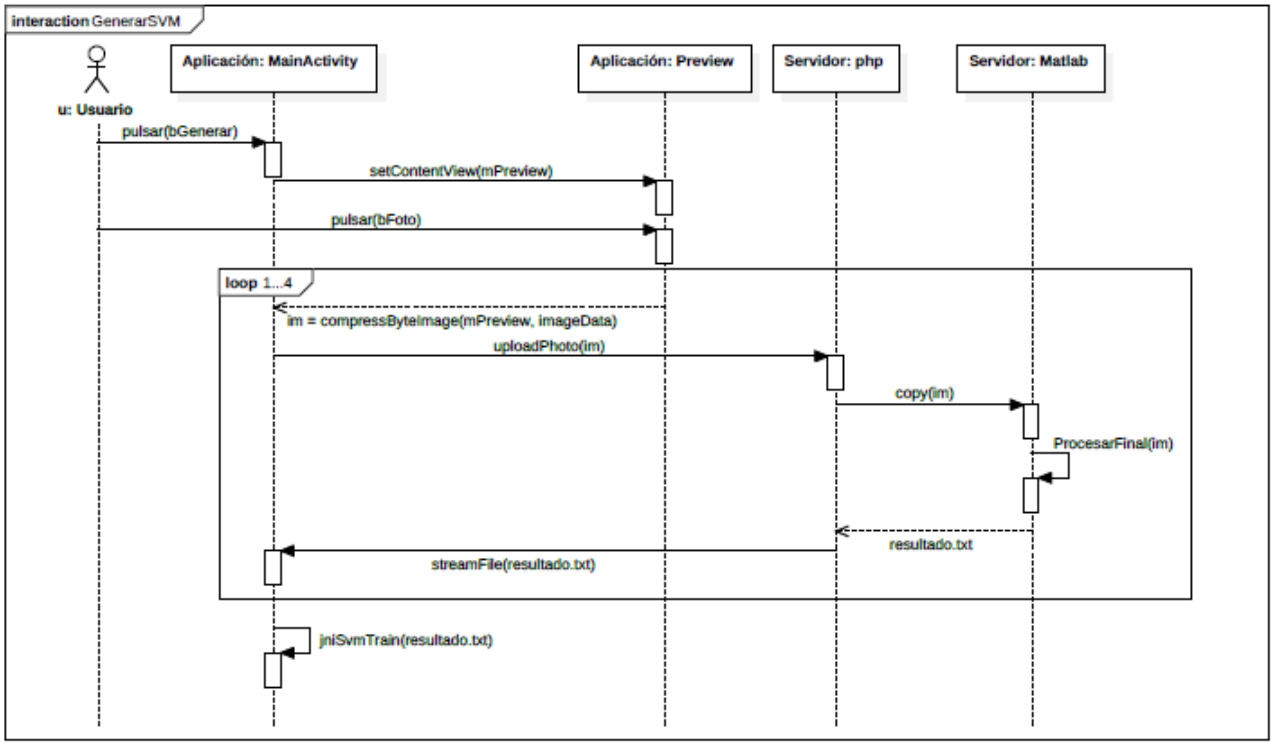
base radial y sigmoide. Se realizarán diferentes pruebas con cada uno de estos para hallar el más eficaz.

- **Otros parámetros:** Incluye una serie de parámetros adicionales para ajustar el SVM pero en nuestro caso se tendrán en cuenta los valores por defecto.

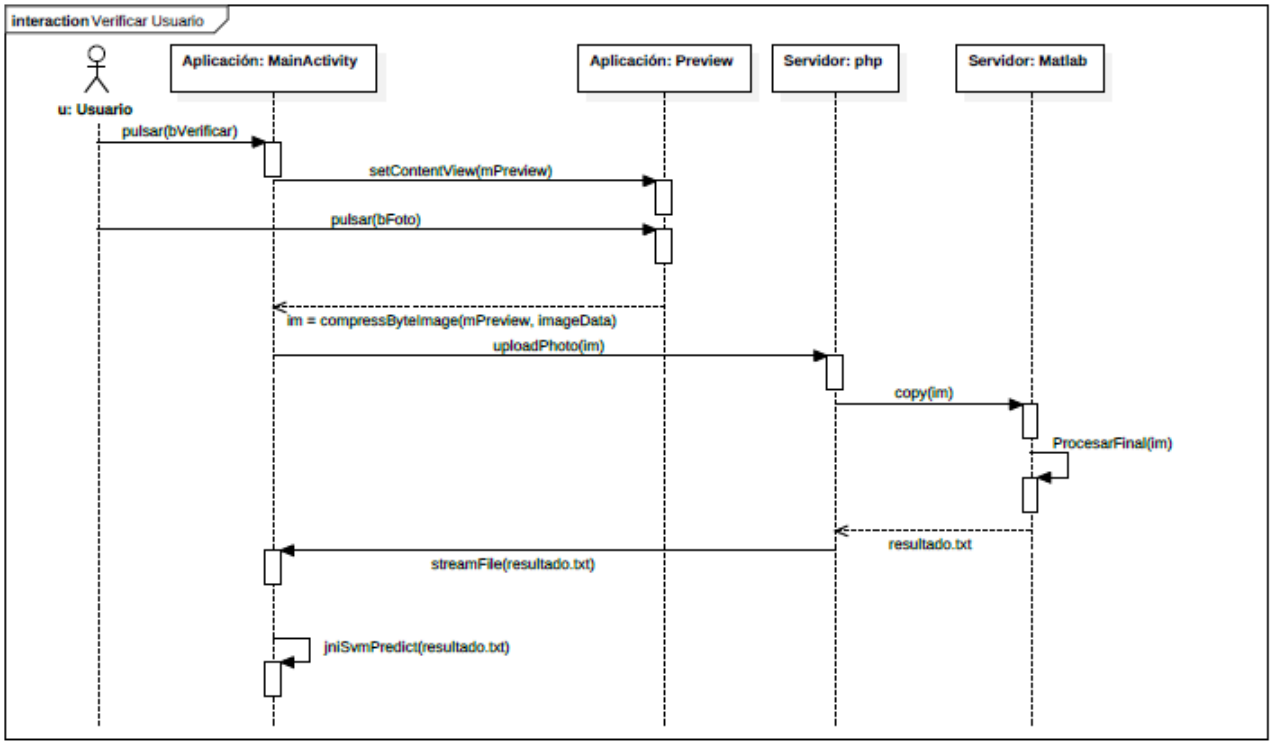
C. ANEXO

Diagrama de secuencia

C.1. Generar las SVM



C.2. Verificar al usuario



D. ANEXO

Código Main Activity Android

```
1  public class MainActivity extends AppCompatActivity {
2
3      static {
4          System.loadLibrary("jnilibsvm");
5      }
6
7      private static final String TAG = "SIFTExampleActivity";
8      private static final int maxFaces = 4;
9      boolean noFace = false;
10     int numFaces = 0;
11     boolean svmNew = false;
12     Preview mPreview;
13     private Context mContext = this;
14
15     private final String SERVERURL = "http://192.168.0.11/svm/vector.php";
16
17     private final static String INPUT_IMG_FILENAME = "/temp.jpg"; //name for storing image
        captured by camera view
18
19     //flag to check if camera is ready for capture
20     private boolean mCameraReadyFlag = true;
21
22     @Override
23     protected void onCreate(Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25
26         //remove the title bar
27         requestWindowFeature(Window.FEATURE_NO_TITLE);
28         inicializar();
29
30     }
```

```
31
32      private void inicializar() {
33          setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
34          setContentView(R.layout.activity_main);
35
36          final Button bGenerar = (Button) findViewById(R.id.bGenerar);
37          final Button bVerificar = (Button) findViewById(R.id.bVerificar);
38          final Button bBorrar = (Button) findViewById(R.id.bBorrar);
39          final TextView tv = (TextView) findViewById(R.id.tvRespuesta);
40
41          bGenerar.setOnClickListener(new View.OnClickListener() {
42              public void onClick(View v) {
43
44                  //make the screen full screen
45                  getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
46                          WindowManager.LayoutParams.FLAG_FULLSCREEN);
47
48                  mPreview = new Preview(mContext);
49
50                  //set Content View as the preview
51                  setContentView(mPreview);
52                  bVerificar.setEnabled(true);
53                  bBorrar.setEnabled(true);
54                  bGenerar.setEnabled(false);
55
56                  tv.setText("");
57              }
58          });
59
60          bBorrar.setOnClickListener(new View.OnClickListener() {
61              public void onClick(View v) {
62
63                  svmNew = true;
64                  numFaces = 0;
65                  File svm = new File(getFilesDir().toString() + "/svm.txt");
66                  File test = new File(getFilesDir().toString() + "/verify.txt");
67                  svm.delete();
68                  test.delete();
69                  bVerificar.setEnabled(false);
70                  bBorrar.setEnabled(false);
71                  bGenerar.setEnabled(true);
72                  tv.setText("");
73              }
74          });
75
76          bVerificar.setOnClickListener(new View.OnClickListener() {
77              public void onClick(View v) {
78
79                  File test = new File(getFilesDir().toString() + "/output.txt");
80                  test.delete();
81                  File verify = new File(getFilesDir().toString() + "/verify.txt");
82                  verify.delete();
```

```
83
84
85              //make the screen full screen
86              getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
87                      WindowManager.LayoutParams.FLAG_FULLSCREEN);
88
89              mPreview = new Preview(mContext);
90
91              //set Content View as the preview
92              setContentView(mPreview);
93              tv.setText("");
94          }
95      });
96
97      File svm = new File(getFilesDir().toString() + "/svm.txt");
98      if (svm.exists()) {
99          numFaces = maxFaces + 1;
100         bGenerar.setEnabled(false);
101         bVerificar.setEnabled(true);
102         bBorrar.setEnabled(true);
103     } else {
104         svmNew = true;
105         numFaces = 0;
106         bBorrar.setEnabled(false);
107         bVerificar.setEnabled(false);
108         bGenerar.setEnabled(true);
109     }
110 }
111
112 private String leerResultado() {
113     try {
114         FileInputStream f = openFileInput("output.txt");
115         InputStreamReader isr = new InputStreamReader(f);
116         BufferedReader bufferedReader = new BufferedReader(isr);
117         StringBuilder sb = new StringBuilder();
118         String line;
119         while ((line = bufferedReader.readLine()) != null) {
120             sb.append(line);
121         }
122         return sb.toString();
123     } catch (FileNotFoundException e) {
124         e.printStackTrace();
125     } catch (IOException e) {
126         e.printStackTrace();
127     }
128     return null;
129 }
130
131 public native void jniSvmTrain(String s);
132
133 public native void jniSvmPredict(String s);
134
```

```
135
136     // Called when shutter is opened
137     Camera.ShutterCallback shutterCallback = new Camera.ShutterCallback() {
138         public void onShutter() {
139         }
140     };
141
142     // Handles data for raw picture
143     Camera.PictureCallback rawCallback = new Camera.PictureCallback() {
144         @Override
145         public void onPictureTaken(byte[] arg0, android.hardware.Camera arg1) {
146         }
147     };
148
149     //store the image as a jpeg image
150     public boolean compressByteImage(Context mContext, byte[] imageData,
151                                      int quality) {
152         File sdCard = Environment.getExternalStorageDirectory();
153         FileOutputStream fileOutputStream = null;
154         int imageWidth, imageHeight, x, y;
155         int numberOfFace = 1;
156         FaceDetector myFaceDetect;
157         FaceDetector.Face[] myFace;
158         float myEyesDistance;
159         int numberOfFaceDetected;
160
161         try {
162             BitmapFactory.Options options = new BitmapFactory.Options();
163             options.inPreferredConfig = Bitmap.Config.RGB_565;
164             options.inSampleSize = 1;    //no downsampling
165             Bitmap myImage = BitmapFactory.decodeByteArray(imageData, 0, imageData.length,
    options);
166
167             Matrix matrix = new Matrix();
168             matrix.postRotate(-90); // anti-clockwise by 90 degrees
169
170             // create a new bitmap from the original using the matrix to transform the result
171             myImage = Bitmap.createBitmap(myImage, 0, 0, myImage.getWidth(), myImage.getHeight(),
     matrix, true);
172
173
174             fileOutputStream = new FileOutputStream(
175                     sdCard.toString() + INPUT_IMG_FILENAME);
176
177             BufferedOutputStream bos = new BufferedOutputStream(
178                     fileOutputStream);
179
180             // pre-process image
181             imageWidth = myImage.getWidth();
182             imageHeight = myImage.getHeight();
183             myFace = new FaceDetector.Face[1];
184             myFaceDetect = new FaceDetector(imageWidth, imageHeight, numberOfFace);
```

```
185             numberOfFaceDetected = myFaceDetect.findFaces(myImage, myFace);
186
187             if (numberOfFaceDetected != 0) {
188                 numFaces++;
189                 noFace = false;
190                 FaceDetector.Face face = myFace[0];
191                 PointF myMidPoint = new PointF();
192                 face.getMidPoint(myMidPoint);
193                 myEyesDistance = face.eyesDistance();
194
195
196                 if ((myMidPoint.x - myEyesDistance) < 0)
197                     x = 0;
198                 else
199                     x = (int) (myMidPoint.x - myEyesDistance);
200
201                 if ((myMidPoint.y - myEyesDistance * 1.5) < 0)
202                     y = 0;
203                 else
204                     y = (int) (myMidPoint.y - myEyesDistance * 1.5);
205
206                 imageWidth = (int) (myMidPoint.x + myEyesDistance - x);
207                 imageHeight = (int) (myMidPoint.y + myEyesDistance * 1.5 - y);
208
209                 Bitmap myFinalFace = createBitmap(myImage, x, y, imageWidth, imageHeight);
210
211
212                 //compress image to jpeg
213                 myFinalFace.compress(Bitmap.CompressFormat.JPEG, quality, bos);
214             } else {
215                 noFace = true;
216                 myImage.compress(Bitmap.CompressFormat.JPEG, quality, bos);
217             }
218
219
220             bos.flush();
221             bos.close();
222             fileOutputStream.close();
223
224         } catch (FileNotFoundException e) {
225             Log.e(TAG, "FileNotFoundException");
226             e.printStackTrace();
227         } catch (IOException e) {
228             Log.e(TAG, "IOException");
229             e.printStackTrace();
230         }
231         return true;
232     }
233
234     // Handles data for jpeg picture
235     Camera.PictureCallback jpegCallback = new Camera.PictureCallback() {
236         @Override
```

```
237         public void onPictureTaken(byte[] imageData, android.hardware.Camera camera) {
238             if (imageData != null) {
239                 Intent mIntent = new Intent();
240                 //compress image
241                 compressByteImage(mContext, imageData, 75);
242                 setResult(0, mIntent);
243 
244                 //** Send image and offload image processing task  to server by starting async
    task **
245                 ServerTask task = new ServerTask();
246                 task.execute(Environment.getExternalStorageDirectory().toString() +
    INPUT_IMG_FILENAME);
247                 if (noFace) camera.startPreview();
248                 else if (numFaces == maxFaces) {
249                     try {
250                         sleep(2250);
251                         inicializar();
252                     } catch (InterruptedException e) {
253                         e.printStackTrace();
254                     }
255                 } else if (numFaces > maxFaces) {
256                     try {
257                         sleep(2270);
258                         inicializar();
259                         TextView tv = (TextView) findViewById(R.id.tvRespuesta);
260                         String result = leerResultado();
261                         if (result != null && result.equals("1")) {
262                             tv.setText("Verificado");
263                         } else if (result != null) {
264                             tv.setText("Usuario incorrecto");
265                         }
266 
267                     } catch (InterruptedException e) {
268                         e.printStackTrace();
269                     }
270 
271                 } else
272                     //start the camera view again.
273                     camera.startPreview();
274             }
275         }
276     };
277 
278     //*********************************************************************************
279     //UI
280     //*********************************************************************************
281 
282     //onKeyDown is used to monitor button pressed and facilitate the switching of views
283     @Override
284     public boolean onKeyDown(int keycode, KeyEvent event) {
285         //check if the camera button is pressed
286         if (keycode == KeyEvent.KEYCODE_CAMERA) {
```

```
287             if (mCameraReadyFlag == true)//switch to camera view
288             {
289                 mCameraReadyFlag = false;
290                 mPreview.camera.takePicture(shutterCallback, rawCallback, jpegCallback);
291             }
292             return true;
293         }
294         return super.onKeyDown(keycode, event);
295     }
296 
297     //**************************************************************************
298     //Push image processing task to server
299     //**************************************************************************
300 
301     public class ServerTask extends AsyncTask<String, Integer, Void> {
302         public byte[] dataToServer;
303 
304         //Task state
305         private final int UPLOADING_PHOTO_STATE = 0;
306         private final int SERVER_PROC_STATE = 1;
307 
308         private ProgressDialog dialog;
309 
310         //upload photo to server
311         HttpURLConnection uploadPhoto(FileInputStream fileInputStream) {
312 
313             final String serverFileName = "test" + (int) Math.round(Math.random() * 100000) + ".
     jpg";
314             final String lineEnd = "\r\n";
315             final String twoHyphens = "--";
316             final String boundary = "*****";
317 
318             try {
319                 URL url = new URL(SERVERURL);
320                 // Open a HTTP connection to the URL
321                 final HttpURLConnection conn = (HttpURLConnection) url.openConnection();
322                 // Allow Inputs
323                 conn.setDoInput(true);
324                 // Allow Outputs
325                 conn.setDoOutput(true);
326                 // Don't use a cached copy.
327                 conn.setUseCaches(false);
328 
329                 // Use a post method.
330                 conn.setRequestMethod("POST");
331                 conn.setRequestProperty("Connection", "Keep-Alive");
332                 conn.setRequestProperty("Content-Type", "multipart/form-data;boundary=" +
     boundary);
333 
334                 DataOutputStream dos = new DataOutputStream(conn.getOutputStream());
335 
336                 dos.writeBytes(twoHyphens + boundary + lineEnd);
```

```
337                 if (noFace)
338                     dos.writeBytes("Content-Disposition: form-data; name=\"uploadedfile\";
    filename=\"" + "noFace.jpg" + "\"" + lineEnd);
339                 else if (numFaces > maxFaces)
340                     dos.writeBytes("Content-Disposition: form-data; name=\"uploadedfile\";
    filename=\"" + "verifyFace.jpg" + "\"" + lineEnd);
341                 else if (svmNew) {
342                     dos.writeBytes("Content-Disposition: form-data; name=\"uploadedfile\";
    filename=\"" + "newSvm.jpg" + "\"" + lineEnd);
343                     svmNew = false;
344                 } else
345                     dos.writeBytes("Content-Disposition: form-data; name=\"uploadedfile\";
    filename=\"" + serverFileName + "\"" + lineEnd);
346                 dos.writeBytes(lineEnd);
347
348                 // create a buffer of maximum size
349                 int bytesAvailable = fileInputStream.available();
350                 int maxBufferSize = 1024;
351                 int bufferSize = Math.min(bytesAvailable, maxBufferSize);
352                 byte[] buffer = new byte[bufferSize];
353
354                 // read file and write it into form...
355                 int bytesRead = fileInputStream.read(buffer, 0, bufferSize);
356
357                 while (bytesRead > 0) {
358                     dos.write(buffer, 0, bufferSize);
359                     bytesAvailable = fileInputStream.available();
360                     bufferSize = Math.min(bytesAvailable, maxBufferSize);
361                     bytesRead = fileInputStream.read(buffer, 0, bufferSize);
362                 }
363
364                 // send multipart form data after file data...
365                 dos.writeBytes(lineEnd);
366                 dos.writeBytes(twoHyphens + boundary + twoHyphens + lineEnd);
367                 publishProgress(SERVER_PROC_STATE);
368                 // close streams
369                 fileInputStream.close();
370                 dos.flush();
371
372                 return conn;
373             } catch (MalformedURLException ex) {
374                 Log.e(TAG, "error: " + ex.getMessage(), ex);
375                 return null;
376             } catch (IOException ioe) {
377                 Log.e(TAG, "error: " + ioe.getMessage(), ioe);
378                 return null;
379             }
380         }
381
382         //get image result from server and display it in result view
383         void getResultImage(HttpURLConnection conn, int numFaces) {
384             // retrieve the response from server
```

```
385             InputStream is;
386             FileOutputStream fos = null;
387             final File s = getFilesDir();
388             try {
389                 is = conn.getInputStream();
390                 //get result image from server
391                 if (noFace)
392                     fos = new FileOutputStream(s.toString() + "/trash.txt");
393                 else if (numFaces == maxFaces)
394                     fos = new FileOutputStream(s.toString() + "/svm.txt");
395                 else if (numFaces > maxFaces)
396                     fos = new FileOutputStream(s.toString() + "/verify.txt");
397                 else
398                     fos = new FileOutputStream(s.toString() + "/trash.txt");
399                 int read = 0;
400                 byte[] bytes = new byte[1024];
401 
402                 while ((read = is.read(bytes)) != -1) {
403                     fos.write(bytes, 0, read);
404                 }
405 
406                 is.close();
407             } catch (IOException e) {
408                 Log.e(TAG, e.toString());
409                 e.printStackTrace();
410             }
411         }
412 
413         //Main code for processing image algorithm on the server
414 
415         void processImage(String inputImageFilePath, int numFace) {
416             publishProgress(UPLOADING_PHOTO_STATE);
417             File inputFile = new File(inputImageFilePath);
418             try {
419 
420                 //create file stream for captured image file
421                 FileInputStream fileInputStream = new FileInputStream(inputFile);
422 
423                 //upload photo
424                 final HttpURLConnection conn = uploadPhoto(fileInputStream);
425 
426                 //get processed photo from server
427                 if (conn != null) {
428                     getResultImage(conn, numFace);
429                 }
430                 fileInputStream.close();
431 
432                 File trash = new File(getFilesDir().toString() + "/trash.txt");
433                 trash.delete();
434                 if (noFace) ;
435                 else if (numFace == maxFaces)
```

```
436                     jniSvmTrain("-t 3 " + getFilesDir().toString() + "/svm.txt " + getFilesDir().
    toString() + "/modelo.model");
437                 else if (numFace > maxFaces)
438                     jniSvmPredict(getFilesDir().toString() + "/verify.txt " + getFilesDir().
    toString() + "/modelo.model " + getFilesDir().toString() + "/output.txt");
439 
440             } catch (FileNotFoundException ex) {
441                 Log.e(TAG, ex.toString());
442             } catch (IOException ex) {
443                 Log.e(TAG, ex.toString());
444             }
445         }
446 
447         public ServerTask() {
448             dialog = new ProgressDialog(mContext);
449         }
450 
451         protected void onPreExecute() {
452             this.dialog.setMessage("Photo captured");
453             this.dialog.show();
454         }
455 
456         @Override
457         protected Void doInBackground(String... params) {            //background operation
458             String uploadFilePath = params[0];
459             processImage(uploadFilePath, numFaces);
460             //release camera when previous image is processed
461             mCameraReadyFlag = true;
462             return null;
463         }
464 
465         //progress update, display dialogs
466         @Override
467         protected void onProgressUpdate(Integer... progress) {
468             if (progress[0] == UPLOADING_PHOTO_STATE) {
469                 dialog.setMessage("Uploading");
470                 dialog.show();
471             } else if (progress[0] == SERVER_PROC_STATE) {
472                 if (dialog.isShowing()) {
473                     dialog.dismiss();
474                 }
475                 dialog.setMessage("Processing");
476                 dialog.show();
477             }
478         }
479 
480         @Override
481         protected void onPostExecute(Void param) {
482             if (dialog.isShowing()) {
483                 dialog.dismiss();
484             }
485         }
```

```
486     }
487 }
```

E. ANEXO

Código de Matlab

E.1. Función de monitorización de archivos

```
function Observer(net)
dir_content = dir('C:\Program Files\MATLAB\MATLAB Production Server\R2015a\toolbox\matconvnet
    -1.0-beta23\matconvnet-1.0-beta23');
filenames = {dir_content.name};
current_files = filenames;
while true
  dir_content = dir('C:\Program Files\MATLAB\MATLAB Production Server\R2015a\toolbox\matconvnet
    -1.0-beta23\matconvnet-1.0-beta23');
  filenames = {dir_content.name};
  new_files = setdiff(filenames,current_files);
  if ~isempty(new_files)
    % deal with the new files
    ProcesarFinal(new_files,net);
  end
  current_files = filenames;
end
end
```

E.2. Función de procesado de imagen

```
function result = ProcesarFinal(input_img_name,net)

output_txt_path =('./SVMs/resultado.txt');

[v,k]=procesar(input_img_name,net);
if strcmp(input_img_name, 'newSvm.jpg')
    delete ./SVMs/resultado.txt
    copyfile ./SVMs/CEntrenamiento.txt ./SVMs/resultado.txt
    CEntrenamiento(v,k,output_txt_path);
    delete ./newSvm.jpg
else if strcmp(input_img_name, 'verifyFace.jpg')
        delete ./SVMs/resultado.txt
        pause(0.01);
        CEntrenamiento(v,k,output_txt_path);
        delete ./verifyFace.jpg
else
    CEntrenamiento(v,k,output_txt_path);
end
end
```

E.3. Función de generación de archivos

```
function CEntrenamiento(v,k,output_txt_path)
fid = fopen(output_txt_path, 'a+t');
[n,s]=size(v);
for i = 1:n
    if(k(i,1) == 'Y')
        fprintf( fid, '%s ', '+1');
    else
        fprintf( fid, '%s ', '-1');
    end
    for j = 1:4096
        if(v(i,j) ~= 0)
            fprintf( fid, '%d:%f ',j ,v(i,j));
        end
    end
    fprintf( fid, '\n');

end
fclose(fid);
```

Bibliografía

[DBW, 2008] (2008). MORPH Non-Commercial Release Whitepaper.

[MCN, 2017] (2017). MatConvNet. http://www.vlfeat.org/matconvnet/.

[MOR, 2017] (2017). MORPH Database (Academic Use Only). $https$: $//ebill.uncw.edu/C20231_{u}stores/web/product_{d}etail.jsp?PRODUCTID = 8$.

[KT, 2017] (2017). Tablero de Control Kanban Online para Empresas | Software de Gestión Visual de Proyectos | Kanban Tool. https://kanbantool.com/es/.

[Anzola, 2015] Anzola, N. S. (2015). Máquinas de soporte vectorial y redes neuronales artificiales en la predicción del movimiento USD/COP spot intradiario.

[Ballesteros,] Ballesteros, A. Tutorial introduccion a las Redes Neuronales. http://www.redes-neuronales.com.es/index.htm.

[Belver, 2016] Belver, C. (2016). Comparative Study of Human Age Estimation Based on Hand-crafted and Deep Face Features.

[Briega, 2016] Briega, R. E. L. (2016). Redes neuronales convolucionales con TensorFlow.

[Chang and Lin, 2000] Chang, C.-C. and Lin, C.-J. (2000). LIBSVM – A Library for Support Vector Machines. http://www.csie.ntu.edu.tw/ cjlin/libsvm/.

[Grady Booch, 2006] Grady Booch, James Rumbaugh, I. J. (2006). *El lenguaje unificado de modelado*. Addison-wesley.

[Gutiérrez,] Gutiérrez, J. M. Introducción a las Redes Neuronales.

[Haykin, 1994] Haykin, S. (1994). *Neural Networks A comprehensive foundation*. Macmillan.

[Laird, 1988] Laird, P. N. J. (1988). *El ordenador y la mente: Introducción a la ciencia cognitiva.* Ed. Paidós.

[Lamport, 1986] Lamport, L. (1986). *LaTeX: User's Guide & Reference Manual*. Addison-Wesley.

[Mallick, 2016] Mallick, S. (2016). Histogram of Oriented Gradients. http://www.learnopencv.com/histogram-of-oriented-gradients/.

[N.V., 2017] N.V., T. (2017). Planificador de proyectos | Diagrama de Gantt Online | Tom's Planner. https://www.tomsplanner.es/.

[Parkhi et al., 2015] Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). Deep Face Recognition.

[Pokharna, 2016] Pokharna, H. (2016). The best explanation of Convolutional Neural Networks on the Internet!

[Roa, 2016] Roa, L. (2016). Analysis of facial expressions in children: Experiments based on the DB 2Child Affective Facial Expression.

[S. Theodoridis, 2009] S. Theodoridis, K. K. (2009). *Pattern recognition.* Academic Press.

[Sathyanarayana, 2014] Sathyanarayana, S. (2014). A Gentle Introduction to Backpropagation.

[Stanford, 2015] Stanford, U. (2015). Tutorial on Client-Server Communications. https://web.stanford.edu/class/ee368/Android/Tutorial-3-Server-Client-Communication-for-Android.pdf.

[Tung, 2015] Tung, Y.-C. (2015). The well-known libSVM on Android layered by Java Native Interface (JNI). https://github.com/yctung/AndroidLibSvm.

[Valveny, a] Valveny, E. Lecture 11 - Support Vector Machines (SVM): Conceptos básicos. https://www.coursera.org/learn/clasificacion-imagenes/lecture/52lRD/support-vector-machines-svm-conceptos-basicos.

[Valveny, b] Valveny, E. Lecture 12 - Support Vector Machines (SVM): Desarrollo matemático. https://www.coursera.org/learn/clasificacion-imagenes/lecture/ztrcv/support-vector-machines-svm-desarrollo-matematico.

[Vedaldi and Lenc, 2015] Vedaldi, A. and Lenc, K. (2015). Matconvnet – convolutional neural networks for matlab. In *Proceeding of the ACM Int. Conf. on Multimedia*.