UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

"DESARROLLO DE UNA APLICACIÓN MÓVIL PARA LE DETECCIÓN Y CLASIFICACIÓN DE HOJAS DE ÁRBOLES"

TRABAJO FINAL DE GRADO

Autor/a:
Jaume Blanco Alambiaga

Tutor/a:
José Ignacio Herranz Herruzo

GANDIA, 2015

Resumen

Con este proyecto se pretende desarrollar una aplicación móvil basada en las técnicas propias de la visión artificial, mediante la cual se puedan realizar fotografías a una hoja determinad y, tras la evaluación de distintas características morfológicas de ésta, poder clasificar la hoja según la especie de árbol a la que pertenezca. Para ello se ha realizado e introducido en la aplicación, una base de datos, fácilmente ampliable, con las seis especies evaluadas para este trabajo.

Palabras clave

OpenCV, Visión Artificial, Clasificación de Hojas, Árbol, Android.

Abstract

This project is aimed to develop a mobile application based on proprietary techniques of artificial vision. With this application we can take pictures of a particular leaf, and after evaluating its different morphological characteristics, we can classify the leaf in the correct species. For this reason a database easily expandable has been included into the application, with six species evaluated for this work.

Keywords

OpenCV, Computer Vision, Android, Leafs classification, Three. Little.

Contenido

1. INTRODUCCIÓN

En la actualidad, los sistemas de visión artificial y de reconocimiento de imágenes están tomando mucha fuerza en el ámbito industrial, doméstico y del entretenimiento, volviéndose prácticamente indispensables en el funcionamiento del día a día de nuestra sociedad. Nos encontramos en la era de mayor desarrollo de esta tecnología, instaurada ya en multitud de cadenas de montaje, de clasificación de objetos, alimentos e incluso seres vivos, en multitud de aplicaciones móviles, PC, etc.

Se están desarrollando multitud de aplicaciones que demandan esta tecnología, utilizándose así tanto con fines médicos, como con fines industriales, para la clasificación de piezas o para la detección de fallos de fabricación, en el mundo de la ciencia…

1.1. Motivación

En numerosas ocasiones, nos hemos encontrado en el monte, el campo o incluso paseando por la ciudad, observando árboles y plantas desconocidas, pero sin los conocimientos necesarios para poder averiguar la especie exacta de ese árbol que tanto nos ha gustado o interesado. Existen multitud de aplicaciones móviles destinadas al reconocimiento facial de textos, códigos de barras, monumentos… Como ya he podido comprobar a posteriori de la decisión de embarcarme en este proyecto, también existen aplicaciones dedicadas al reconocimiento y clasificación de hojas, pero éstas, en su mayoría, no se centran en plantas y árboles de nuestro territorio.

La idea surge tras el descubrimiento de una aplicación, de ámbito estatal, que permite la clasificación de plantas de nuestro territorio a partir de un cuestionario que va avanzando y acotando el número de posibilidades, hasta conducirnos a las distintas soluciones posibles. Una vez observada esta aplicación, se genera una duda, ¿Por qué no ampliarla mediante un sistema de reconocimiento digital de imágenes?

1.2. Estado del arte

Actualmente, como ya se ha expuesto anteriormente, podemos encontrar infinidad de aplicaciones basadas en el reconocimiento de imágenes para la obtención de información. Estas aplicaciones, son muy variadas, podemos encontrar aplicaciones dedicadas al reconocimiento óptico de caracteres (OCR), como Cam Scanner o “TextGrabber + Translator” que incluso nos permite traducir textos de más de 60 idiomas, al reconocimiento facial, instauradas ya en cualquier cámara de video y/o fotos, lectores de códigos de barras y QR, aplicaciones que nos permiten fotografiar nuestra ropa favorita para encontrar otra similar a nuestros gustos (Stylethief)…

El gran gigante Google ha desarrollado una aplicación, Goggles, que podría englobar todas las anteriormente nombradas, ya que, a través de una foto cualquiera es capaz

de segmentarla en objetos que ella misma reconoce y aportarnos información acerca de éstos o mostrarnos imágenes similares. Muy útil para el reconocimiento de textos, obras de arte, monumentos, etc.

1.2.1. Apps similares

Como en el resto de ámbitos, también en el referente a la botánica, se ha fijado en esta tecnología, y ha querido desarrollarla en su propio beneficio. De esta forma, cabe destacar las siguientes aplicaciones, que de una forma u otra, son similares al trabajo que este documento viene a exponer.

- **LeafSnap:** serie de guías electrónicas de campo desarrolladas por investigadores de la Universidad de Columbia, la Universidad de Maryland, y la Smithsonian Institution. Esta aplicación de móvil gratuita utiliza el software de reconocimiento visual para ayudar a identificar las especies de árboles a partir de fotografías de sus hojas. Leafsnap incluye actualmente los árboles que se encuentran en el noreste de Estados Unidos y Canadá, y pronto crecerá para incluir los árboles de todo el territorio continental de Estados Unidos. Esta aplicación está disponible sólo para iPhone, motivo por el cual no ha sido posible probarla antes.

- **Pl@ntNet**: aplicación para la recopilación, anotación y recuperación de imágenes que ayuda en la identificación de plantas. Fue desarrollada por un consorcio formado por científicos de CIRAD (Centro de Cooperación Internacional en Investigación Agronómica para el Desarrollo), INRA (Instituto Nacional para la Investigación Agronómica), INRIA (Instituto Nacional de Investigación en Informática y Automática), IRD (Instituto de Investigación para el Desarrollo), y la red de botánicos francófonos Tela Botánica en virtud de un proyecto financiado por la Fundación Agrópolis. Se incluye un sistema de apoyo para la identificación automática de las plantas a partir de imágenes que se comparan con otras, almacenadas en una base de datos. Esta aplicación funciona con más de 4.100 especies de plantas silvestres de la flora francesa metropolitana, cuya lista está disponible a través de la aplicación. El nombre de especies procesadas y el número de imágenes utilizadas evolucionan con las contribuciones de los usuarios al proyecto, de forma que cuando se identifica correctamente una especie, se puede participar en el proyecto transmitiendo su observación con el botón de "contribución". Estas contribuciones son sujetas a un proceso de validación. La aplicación funciona mejor cuando las fotos presentadas se centran en un órgano o una parte precisa de la planta, como por ejemplo, fotografía de hojas de árboles sobre un fondo uniforme.

- **ArbolApp:** aplicación basada en la investigación del Real Jardín Botánico del CSIC que ayuda a identificar los árboles silvestres de la Península Ibérica y las Islas Baleares. Esta aplicación incluye 118 especies de árboles autóctonos y los más frecuentemente asilvestrados en Andorra, Portugal continental, España peninsular y las Islas Baleares. Esta aplicación, a diferencia de las dos anteriores, no funciona a través de la detección digital de imágenes si no que

se ha desarrollado con 2 tipos de búsqueda (guiada y abierta) entre las que el usuario puede elegir libremente para identificar especies de manera intuitiva. Los contenidos están orientados a todas aquellas personas que deseen iniciarse o profundizar en el conocimiento de los árboles de su entorno. Por ello, se ha hecho un esfuerzo por utilizar un lenguaje asequible y explicaciones sencillas sin abandonar el rigor científico. Pero el gran problema de este tipo de aplicaciones, es la gran cantidad de espacio que ocupa, en este caso en concreto, hablamos de 45MB, a diferencia de la pl@ntNet que tan sólo ocupa 13MB.

2. DESARROLLO DEL PROYECTO

En este punto se tratará de explicar todo el material requerido para el desarrollo del proyecto, con la finalidad de esclarecer el porqué de la elección de éstos. Se describirá tanto el equipo requerido, necesario para la instalación de programas y la simulación de nuestra aplicación, como el software utilizado durante la programación de ésta.

2.1. Equipo utilizado

Se han utilizado, un ordenador, que al menos tenga los requisitos mínimos que exige Android Studio para el uso eficiente de este software, que viene siendo:

- Sistema operativo Windows (8/7/Vista/2003 de 32 o 64 bits), MAC OS X (10.8.5 o superior, hasta 10.9 (Mavericks)) o Linux.
- Un mínimo de 2GB de memoria RAM, aunque se recomienda 4 GB.
- 400 MB de espacio en el disco duro.
- Una resolución de pantalla mínima de 1200x800.
- Tener instalado Java Development Kit (JDK) 7.
- Necesita de al menos 1 GB para Android SDK, emulador de imágenes del sistema, y cachés.

Para la emulación e instalación de nuestra aplicación, también se ha hecho uso de un Smartphone con un sistema operativo Android y cámara incorporada. En este caso se ha utilizado el Smartphone Alcatel One Touche Pop-5 con la versión de Android 4.4.2.

2.2. Software requerido

Para el desarrollo de la aplicación destacaremos el uso del compilador Android Studio, necesario para la programación y diseño tanto de la interface de ésta, como del algoritmo necesario para la detección y clasificación de nuestras imágenes, para el que ha sido necesaria la instalación de la librería OpenCV.

2.2.1. Android Studio

Android Studio es un entorno de desarrollo (IDE), basado en IntelliJ IDEA de la compañía JetBrains para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, con el objetivo de crear un entorno **dedicado en exclusiva a la programación de aplicaciones** para dispositivos Android. Reemplazó a Eclipse como el IDE oficial para el desarrollo de éstas, que hasta la fecha había sido acogido con éxito por todos los desarrolladores, pero presentaba problemas debido a su lentitud en el desarrollo de versiones que solucionaran las carencias actuales (es indispensable recordar que Eclipse es una plataforma de desarrollo, diseñada para ser

extendida a través de plugins). La migración de Android Studio a Eclipse se está realizando con lentitud.

Android Studio utiliza una licencia de software libre Apache 2.0, está programado en Java y es multiplataforma. La primera versión estable fue publicada en diciembre de 2014, disponible para las plataformas Microsoft Windows, Mac OS X y GNU/Linux. Se puede descargar de forma completamente gratuita desde la plataforma de Android https://developer.android.com/sdk/index.html y su instalación y configuración es muy sencilla.

Las principales características que incluye Android Studio son:

- Soporte para programar aplicaciones para Android Wear (sistema operativo para dispositivos corporales como por ejemplo un reloj).
- Herramientas Lint (detecta código no compatible entre arquitecturas diferentes o código confuso que no es capaz de controlar el compilador) para detectar problemas de rendimiento, usabilidad y compatibilidad de versiones.
- Integración de la herramienta Gradle encargada de gestionar y automatizar la construcción de proyectos, como pueden ser las tareas de testing, compilación o empaquetado.
- Nuevo diseño del editor con soporte para la edición de temas.
- Nueva interfaz específica para el desarrollo en Android.
- Permite la importación de proyectos realizados en el entorno Eclipse, que a diferencia de Android Studio (Gradle) utiliza ANT.
- Posibilita el control de versiones accediendo a un repositorio desde el que poder descargar Mercurial, Git, Github o Subversion.
- Alertas en tiempo real de errores sintácticos, compatibilidad o rendimiento antes de compilar la aplicación.
- Vista previa en diferentes dispositivos y resoluciones.
- Integración con Google Cloud Platform, para el acceso a los diferentes servicios que proporciona Google en la nube.
- Editor de diseño que muestra una vista previa de los cambios realizados directamente en el archivo xml.

La elección final de este entorno de desarrollo no ha sido fundamentada en datos concretos y específicos, ya que se han encontrado argumentos favorables para ambos casos. Por lo tanto, esta decisión ha sido fundamentada teniendo en cuenta que este entorno tiene perspectivas de desbancar a Eclipse en el ámbito del desarrollo de aplicaciones, ya que al ser adoptado por Google como IDE oficial, todo hace suponer que las actualizaciones futuras se centrarán en éste. Por tanto se ha creído conveniente desarrollar la aplicación que nos ocupa, en el entorno de trabajo de Android Studio, y familiarizarnos con él para un posible uso profesional en el futuro.

2.2.2. Biblioteca OpenCV

OpenCV es un conjunto de bibliotecas de código abierto u Open Source bajo licencia BSD (es una licencia de software libre permisiva en comparación con otras, estando muy cercana al dominio público, que permite el uso del código fuente en software no libre) desarrolladas en un principio por Intel y disponibles desde 1999. La biblioteca está escrita en C y C++ y se pueden ejecutar desde diversos sistemas operativos como GNU/Linux, Windows y Mac OS X. También existe un desarrollo de interfaces para otras lenguas como Python, Ruby, Matlab,, java, etc.

OpenCV fue diseñado ofreciendo un código muy eficiente y con un fuerte enfoque a aplicaciones capaces de ejecutarse en tiempo real. Puede tomar ventaja de los procesadores multi-núcleo. Uno de los objetivos de OpenCV es proporcionar una infraestructura de visión artificial fácil de usar que ayude a las personas a construir rápidamente aplicaciones bastante sofisticadas. Para eso, la biblioteca contiene más de 500 funciones que abarcan muchas áreas.

Esta biblioteca está dividida en varios módulos integrados, muy potentes y lo suficientemente versátiles para resolver la mayoría de los problemas de visión artificial, para los que están disponibles toda clase de soluciones bien establecidas.

Se pueden recortar imágenes, mejorarlas mediante la modificación de brillo, nitidez y contraste, detectar formas, segmentar imágenes en regiones intuitivamente obvias, detectar objetos en movimiento a tiempo real, reconocer objetos conocidos, estimar el movimiento de un robot, y el uso de cámaras estéreo para obtener una visión 3D del mundo, por nombrar sólo unas pocas aplicaciones. Cabe destacar que estos módulos, están altamente optimizados, diseñados para aplicaciones en tiempo real y para ser ejecutados de forma muy eficaz. Pueden ser consultados en la Tabla 1.

Para la implementación del algoritmo de nuestra aplicación, no han sido utilizados todos los módulos integrados de OpenCV, tan sólo han sido necesarios 3 de estos: Core, Imgproc y Highgui.

	Funcionalidad del módulo
Core	Estructuras de datos, tipos de datos, y gestión de la memoria.
Imgproc	Filtrado de imágenes, transformaciones geométricas de imágenes, estructura y análisis de la forma.
Highgui	Interfaz gráfica de usuario, leer y escribir imágenes y vídeo.
Vídeo	Análisis de movimiento y seguimiento de objetos en el vídeo.
Calib3d	Calibración de la cámara y la reconstrucción 3D desde múltiples puntos de vista.
Features2d	Característica de la extracción, descripción, y juego.
Objdetect	Detección de objetos utilizando cascada y clasificadores de histograma de gradiente.
ML	Modelos estadísticos y de clasificación de algoritmos para su uso en aplicaciones de visión de computadora.
Flann	Biblioteca rápida de aproximación a los vecinos más cercanos, búsquedas rápidas en espacios de alta dimensión (característica).
GPU	Paralelización de algoritmos seleccionados para una rápida ejecución de las GPU.
Stitching	Deformación, mezcla, y el ajuste de paquetes de unión de la imagen.
Nonfree	Implementaciones de algoritmos que están patentados en algunos países.

Tabla 1: Módulos principales de la biblioteca OpenCV

3. DESCRIPCIÓN DEL ALGORITMO EMPLEADO

Una vez descrito el equipo del que se ha hecho uso para el desarrollo de nuestra aplicación, se procede, en este capítulo, a describir el algoritmo implementado para conseguir una eficiente detección y clasificación de la hoja, tras la previa obtención de la imagen, así como el diseño de la interfaz gráfica que nos permitirá observar el proceso y las soluciones.

El algoritmo desarrollado para esta aplicación sigue el formato típico de los sistemas de reconocimiento de imágenes, el cual está dividido en varias etapas, tal y como podemos observar en el siguiente esquema:

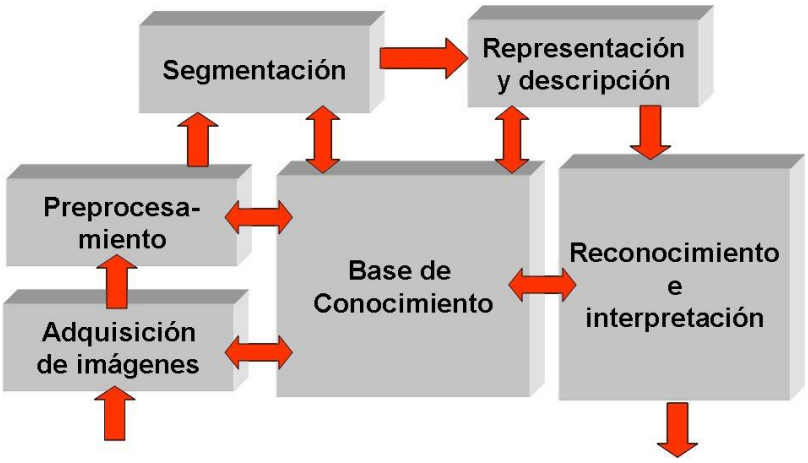


Ilustración 1: Esquema típico de un sistema de visión artificial

Las etapas que caracterizan este tipo de sistemas de visión artificial se componen de:

1. **Adquisición de la imagen:** esta fase es la encargada de obtener la información lumínica de una escena mediante el uso de una cámara fotográfica (cómo en nuestro caso) o de video.

2. **Preprocesamiento:** procesado previo de la imagen adquirida, para la corrección de errores de iluminación, ruido, perspectiva… de modo que se facilite el correcto funcionamiento del resto de etapas del proceso.

3. **Segmentación:** una de las fases principales y más delicadas del proceso, y en la que más tiempo se ha invertido, ya que de esta dependerá la detección correcta del objeto. En este punto se divide la imagen en segmentos y/o contornos según la información contenida en cada pixel.

4. **Representación y descripción:** cálculo de los diferentes parámetros extraídos de los segmentos de la imagen, para el posterior reconocimiento.

5. **Reconocimiento e interpretación:** fase final del proceso en el que, mediante la comparación de los datos anteriores con otros almacenados previamente o mediante la interpretación de estos a través de otros procesos, se toman las decisiones oportunas y se muestran los resultados.

Estas etapas deben estar siempre precedidas de un conocimiento previo del problema que nos permita la descripción y conocimiento de todas ellas.

3.1. Interface gráfica de la app

Se debe resaltar que el peso de este trabajo se ha centrado en el desarrollo de un algoritmo eficiente para la detección y clasificación de hojas, por lo que el diseño de la interfaz no ha sido una prioridad, ya que este compete a otro ámbito, como puede ser a diseñadores gráficos, informáticos u otros. Por este motivo se ha diseñado una interfaz gráfica muy sencilla que, además de ser intuitiva y de fácil manejo para el usuario, también nos ha permitido la visualización correcta de todos los pasos que se han ido dando en el desarrollo de la aplicación.

Una aplicación Android se caracteriza por estar dividida en Activities que, se podría decir, que son las diferentes ventanas por las que se navega a través de una aplicación móvil.

Los componentes principales que pueden formar parte de una aplicación Android son:

- **Activity:** representan el componente principal de la interfaz gráfica de una aplicación Android. Se puede pensar en una actividad como el elemento análogo a una ventana o pantalla en cualquier otro lenguaje visual.

- **View:** son los componentes básicos con los que se construye la interfaz gráfica de la aplicación, análoga, por ejemplo, a los controles de Java o .NET. De inicio, Android pone a nuestra disposición una gran cantidad de controles básicos, como cuadros de texto, botones, listas desplegables o imágenes, aunque también existe la posibilidad de extender la funcionalidad de estos controles básicos o crear nuestros propios controles personalizados.

- **Service:** son componentes sin interfaz gráfica que se ejecutan en segundo plano. En concepto, son similares a los servicios presentes en cualquier otro sistema operativo. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo: actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales (actividades) si se necesita en algún momento la interacción con el usuario.

- **Content Provider:** es el mecanismo que se ha definido en Android para compartir datos entre aplicaciones. Mediante estos componentes es posible compartir determinados datos de nuestra aplicación sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, nuestra aplicación podrá acceder a los datos de otra a través de los Content Provider que se hayan definido.

- **Broadcast Receiver:** es un componente destinado a detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema (por ejemplo: “Batería baja”, “SMS recibido”, “Tarjeta SD insertada”, …) o por otras aplicaciones (cualquier aplicación puede generar mensajes broadcast, es decir, no dirigidos a una aplicación concreta sino a cualquiera que quiera escucharlo).

- **Widget:** Los widgets son elementos visuales, normalmente interactivos, que pueden mostrarse en la pantalla principal (home screen) del dispositivo Android y recibir actualizaciones periódicas. Permiten mostrar información de la aplicación al usuario directamente sobre la pantalla principal.

- **Intent:** Un intent es el elemento básico de comunicación entre los distintos componentes Android que hemos descrito anteriormente. Se pueden entender como los mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un Intent se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, etc.

Por tanto, nuestra aplicación se ha dividido en cuatro actividades diferentes: MainActivity, CameraActivity, LabActivity y HojaActivity. Se podrá navegar fácilmente entre ellas, pudiendo retroceder en cualquier momento. A continuación se describen cada una individualmente.

3.1.1. MainActivity

Ilustración 2: Ventana principal de la aplicación, MainActivity

MainActivity es la ventana de inicio de nuestra aplicación, en ésta se puede escoger entre tomar una fotografía nueva o escoger una imagen almacenada previamente en la galería de nuestro dispositivo móvil.

Al elegir la primera acción, la aplicación nos direccionará directamente al siguiente Activity que se comunicará con la cámara del dispositivo, mientras que si escogemos la segunda opción, se nos abrirá un submenú para escoger el lugar de donde elegiremos la fotografía. Dependiendo de la opción escogida, la aplicación nos redireccionará hacia activities diferentes. Por tanto, si escogemos la primera opción se abrirá directamente la ventana de CameraActivity, mientras que si se escoge la segunda opción, una vez escogida la imagen, está se nos mostrará en la ventana de LabActivity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_arbol_app);

    Log.d(TAG, "ArbolApp.onCreate() : principi ");

    imageView = (ImageView) findViewById(R.id.setPicture);
    CamaraButton = (ImageButton) findViewById(R.id.CamaraButton);
    CamaraButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            startActivity(new Intent(ArbolApp.this,
CameraActivity.class));

        }
    });

    ArchivoButton = (ImageButton) findViewById(R.id.ArchivoButton);
    ArchivoButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            Intent intent = new Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
            intent.setType("image/");
            startActivityForResult(intent.createChooser(intent,
"Escoge una app de imagen:"), SELECT_PICTURE);
        }
    });
}
```

3.1.2. CameraActivity

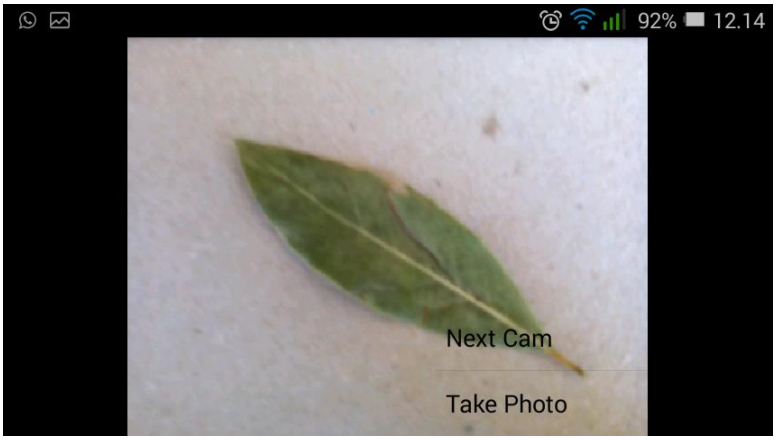


Ilustración 3: Vista de CameraActivity

En este Activivity, la aplicación se comunica con la cámara del dispositivo, pero no a través de una aplicación externa de Android, si no, abriendo la cámara directamente desde nuestra aplicación y tomando la foto en el formato adecuado para que podamos trabajar con ella mediante la biblioteca OpenCV.

Una vez capturada la imagen pulsando sobre “Take Photo”, esta es almacenada en la carpeta de la aplicación, la cual se creará en caso de no haberlo hecho todavía. Esta acción, que a priori es muy sencilla puesto que, cuando programamoas para PC, tanto en C++ como en java, se puede hacer con tan solo una función, Highgui.imwrite(String namePhoto, Mat img), cuando programamos para Android se vuelve más complejo, puesto que se debe determinar la ruta y los metadatos de la foto, crearla e insertarla en MediaStore, tal y cómo se muestra en el código siguiente.

```
private void takePhoto(final Mat rgba) {
// Determine the path and metadata for the photo.
        final long currentTimeMillis = System.currentTimeMillis();
        final String appName = getString(R.string.app_name);
        final String galleryPath =
                Environment.getExternalStoragePublicDirectory(
                        Environment.DIRECTORY_PICTURES).toString();
        final String albumPath = galleryPath + "/" + appName;
        final String photoPath = albumPath + "/" +
                currentTimeMillis + ".png";
        final ContentValues values = new ContentValues();     …

       …
// Ensure that the album directory exists.
        File album = new File(albumPath);
        if (!album.isDirectory() && !album.mkdirs()) {
            Log.d(ArbolApp.TAG, "Failed to create album directory at "
+ albumPath);
            onTakePhotoFailed();
            return;
        }
// Try to create the photo.
        Imgproc.cvtColor(rgba, mBgr, Imgproc.COLOR_RGBA2BGR, 3);
        Highgui.imwrite(photoPath, mBgr)

// Try to insert the photo into the MediaStore.
        Uri uri;
        try {
            uri = getContentResolver().insert(
                    MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
values);
        } catch (final Exception e) {
            Log.d(ArbolApp.TAG, "Failed to insert photo into
MediaStore");
            e.printStackTrace();
            return;
        }

    }
```

Una vez guardada la imagen, se mostrará automáticamente en la ventana de LabActivity, dónde comenzaremos los procesos de detección y clasificación de la hoja.

3.1.3. LabActivity

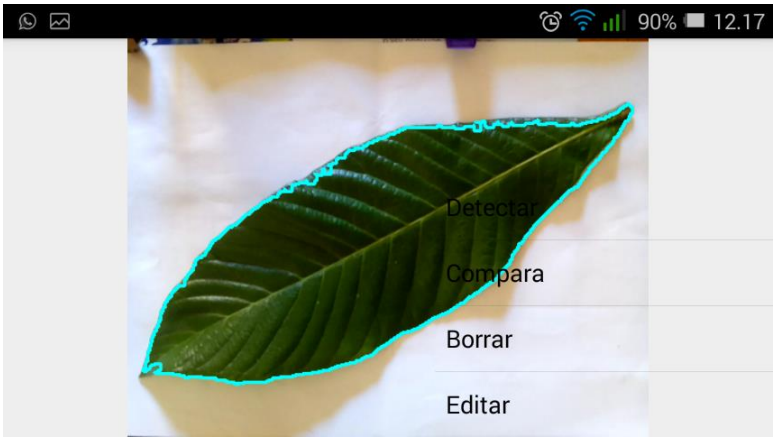


Ilustración 4: Ventana de la actividad LabActivity

Este Activity ha sido creado como paso intermedio para poder ir viendo, paso a paso, el correcto funcionamiento de las diferentes funciones, necesarias para la buena detección de la hoja, que se explicarán más adelante. En este apartado, se ha desarrollado todo el algoritmo de detección y clasificación de la hoja.

Se ha diseñado un menú desplegable con las opciones Editar, Borrar, Detectar y Comparar, que desempeñan funciones diferentes.

- **Editar:** editar la imagen con apps externas, de forma que se puedan corregir ciertas carencias de la imagen debidas a la mala iluminación u otros factores.
- **Borrar:** eliminar la imagen.
- **Detectar:** con esta opción la aplicación lleva a cabo las funciones necesarias para la correcta selección de la hoja. Una vez detectada nos muestra por pantalla la misma imagen pero con el contorno de la hoja dibujado, de forma que, si este no es correcto, podamos eliminar la imagen y volver a tomar otra fotografía.
- **Comparar:** una vez realizada la detección ya podemos escoger esta opción, que nos calculará los datos necesarios, hará las comparaciones entre todas las hojas almacenadas y nos redireccionará automáticamente al siguiente activity, HojaActivity.

En un diseño más avanzado de la aplicación, podríamos prescindir de la opción “comparar”, realizando las funciones necesarias en CameraActivity y mostrando directamente la imagen con el contorno de la hoja. Incluso podríamos omitir este activity por completo y redireccionar directamente de CameraActivity a HojaActivity. No obstante, se ha creído conveniente poder observar si la detección es correcta o no (algo que no hace ninguna de las aplicaciones anteriormente nombradas), ya que al tratarse de una aplicación móvil y no poder controlar la iluminación de la escena, no siempre podremos hacer una correcta detección y por tanto la clasificación de la hoja puede ser incorrecta debido a esto.

3.1.4. HojaActivity

Finalmente se llega a la última ventana de la aplicación dónde se mostrará al usuario, cual es el resultado de la búsqueda y se intentará introducir información referente a la especie de árbol deducida, así como imágenes de este y de sus hojas. Se podrá así comparar y tomar la decisión de si el resultado es correcto o no.

También se incorporará la posibilidad de redireccionar a la página de Wikipedia correspondiente a la especie a tratar, de forma que el usuario pueda encontrar más información en la red, y reducir así, la cantidad de información almacenada necesariamente en nuestra app de manera que esta ocupe el mínimo espacio posible.

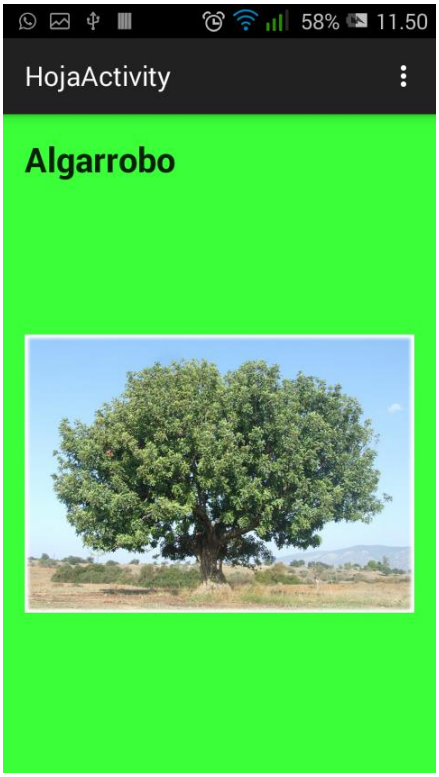


Ilustración 5: Última ventana de la aplicación, HojaActivity

3.2. Reconocimiento de la hoja

Llegados a este punto, vamos a tratar de explicar cuál ha sido el proceso utilizado para llevar a cabo la segmentación de la imagen y descarte de aquellos segmentos que no son de nuestro interés. Para ello se han seguido los siguientes pasos:

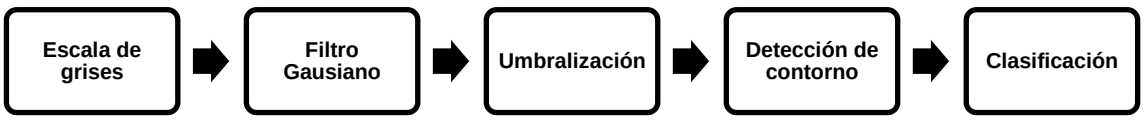


Ilustración 6: Diagrama de bloques del algoritmo de detección de hojas

Una vez cargada la imagen y almacenada en una variable tipo Mat en formato BGR, ésta es desaturada y convertida en una imagen de escala de grises, a través de la función cvtColor. Función mediante la que se pueden transformar las imágenes, fácilmente, a otros modelos de color.

```
Imgproc.cvtColor(Mat src, Mat dst, Imgproc.COLOR_BGR2GRAY);
```

Después, se le apllica un filtro gaussiano, reduciendo así, el número de posibles errores derivados de la presencia de ruido en la imagen original. Este proceso se realiza mediante la función:

```
Imgproc.GaussianBlur(Mat src, Mat dst, Size Ksize, double sigmaX);
```

3.2.1. Umbralización

Este es, quizás, uno de los puntos más delicados de la aplicación, puesto que de él dependerá la correcta clasificación de la hoja fotografiada. Para llevar a cabo este proceso, OpenCV pone a nuestra disposición una serie de funciones que se han evaluado durante el desarrollo de este trabajo y cuya finalidad es obtener una imagen binaria. Es decir, una imagen donde los píxeles correspondientes al fondo tengan el valor 0 (negro) y los píxeles correspondientes a la hoja tengan el valor 255 (blanco):

- **Umbralización simple:** si el valor del pixel es mayor que el valor umbral establecido, se le asigna un valor entre 0 y 255 (normalmente suele ser blanco o negro) y a aquellos píxeles que estén por debajo del umbral, se le asignará el valor contrario. La función utilizada es:

  ```
  Imgproc.threshold (Mat src, Mat dst, double thresh,
                     double maxval, int Type);
  ```

  A parte de introducir las variables correspondientes a la imagen de entrada y la variable donde se almacenará la imagen binaria resultante, se deben introducir los parámetros de valor umbral, utilizado para clasificar los valores de píxel, la variable Maxval que representa el valor que se le dará al píxel si cumple la relación y finalmente el tipo de umbralización. En este caso se ha optado por escoger el tipo de umbralización THRESH_BINARY_INV, que nos dará como resultado una binaria con el fondo negro y la hoja blanca.

- **Umbralización adaptativa:** a diferencia de la función anterior, que utiliza un valor global como valor umbral para toda la imagen, adaptativeThreshold utiliza un algoritmo para calcular un umbral diferente para cada punto de la imagen. Así obtenemos diferentes umbrales para las diferentes regiones de la misma imagen, consiguiendo mejores resultados en imágenes con diferencias de iluminación.

  ```
  Imgproc.adaptiveThreshold(Mat src, Mat dst, double maxval, int
    adaptativeMethod, int thresholdType, int blockSize, int C);
  ```

Cabe destacar tres parámetros importantes:

- adaptativeMethod: Decide cómo se calcula el valor umbral. Esxisten dos métodos diferentes, ADAPTIVE_THRESH_MEAN_C donde se consigue aislar los píxeles de la imagen cuyos valores disten de la media global un cierto valor C constante, y ADAPTIVE_THRESH_ GAUSSIAN_C donde, a diferencia del método anterior, el valor umbral se establece mediante la suma ponderada de los píxeles vecinos.

- Tamaño de bloque: Se decide el tamaño del área de vecindad.

- C: constante que indica cuanto debe desviarse el valor del píxel de la media para ser aislado.

- **Detector de bordes Canny:** La función encuentra los bordes de la imagen de entrada y los marca en la imagen de salida utilizando el algoritmo de Canny. Este algoritmo es un operador desarrollado por John F. Canny en 1986 que utiliza un algoritmo de múltiples etapas para detectar una amplia gama de bordes en imágenes.

```
Canny(Mat img, Mat edge, double thresh1, double thresh2);
```

El valor más pequeño entre threshold1 y threshold2 se utiliza para la vinculación de los bordes, mientras que el valor más grande se utiliza para encontrar los segmentos iniciales de los bordes fuertes.

- **inRange:** Comprueba si los elementos de una matriz se encuentran entre los valores introducidos en dos matrices. La función comprueba el rango de la siguiente manera:

$$dst(I) = lowerb(I)_0 \leq src(I)_0 \leq upperb(I)_0$$

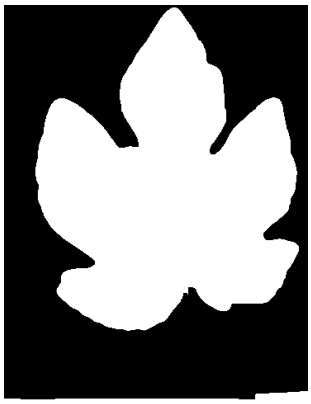
Es decir, que los valores de dst(I) se establecen en 255 si el valor examinado src(I) está dentro de los valores especificados en los escalares lowerb y upperb. La función es:

```
Core.inRange(Mat src, Scalar lowerb, Scalar upperb, Mat dst)
```

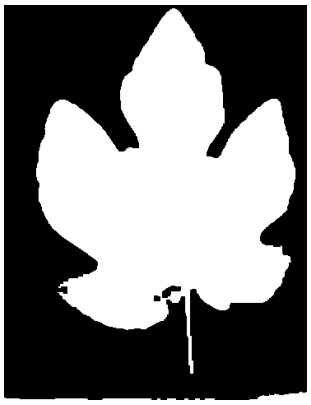
Esta función se ha utilizado con una imagen en formato HSV y se ha evaluado la matriz correspondiente a la saturación, de forma que podamos eliminar lo máximo posible la sombra que se pueda proyectar sobre el fondo de la imagen.

Una vez examinadas y valoradas estas funciones se ha tomado la decisión de aplicar **`adaptativeThreshold()`** y **`inRange()`** a la imagen original, y unir las imágenes resultantes mediante una operación lógica and, de forma que sólo aquellos píxeles que en ambas imágenes tengan el valor 1, mantendrán su valor, mientras que el resto de píxeles se pondrán a 0.

7.a: Imagen original



7.b: adaptativeThreshold con un tamaño de ventana de 1025 píxeles



7.c: inRange

7.d: Operación and de ambas imágenes

Ilustración 7: Proceso de segmentación de una hoja

3.2.2. Detección del contorno

Llegados a este punto del reconocimiento, sólo nos queda detectar el contorno de la hoja, lo cual se realiza mediante una sencilla función, `findContours()`, que nos devuelve una matriz de contornos. Para reducir el número de contornos y así, también la posibilidad de error en la detección, aplicaremos la operación morfológica "close", que eliminará los huecos negros que nos quedan dentro del área detectada.

Finalmente, deberemos escoger cuál es el contorno perteneciente a la hoja, por lo que supondremos que ésta será el motivo principal de la imagen, y por tanto, el mayor contorno detectado, será el correspondiente a la hoja. Esto se realiza fácilmente mediante un bucle for, comparando las áreas calculadas de cada contorno.

Ilustración 8: Contorno detectado de una hoja

3.3. Clasificación

Finalmente, sólo nos queda averiguar a qué especie pertenece la hoja fotografiada. Para ello, deberemos calcular una serie de parámetros derivados del contorno que hemos seleccionado y, por último, compararlos con unas bases de datos, creadas previamente con las medias y/o medianas de estos, para averiguar cuál es el que más se aproxima.

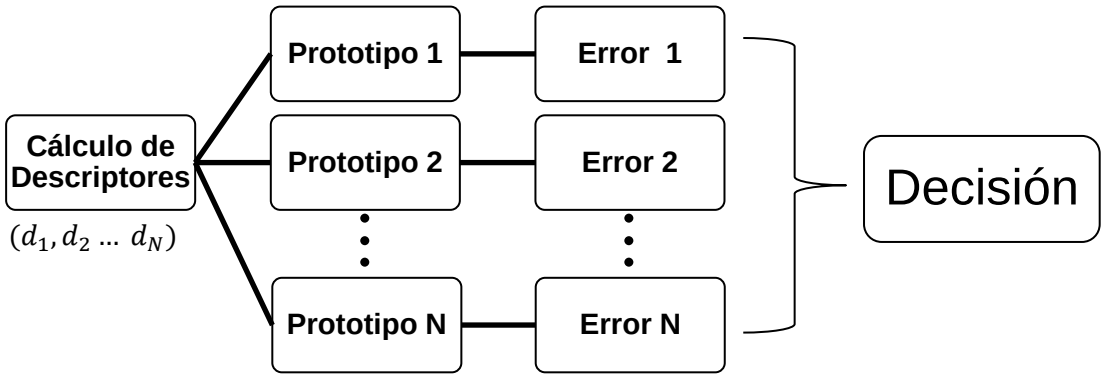


Ilustración 9: Esquema del proceso de clasificación de una hoja

La aplicación se ha desarrollado para diferenciar entre seis especies de árboles diferentes, Ceratonia siliqua (algarrobo europeo), Eriobotrya japónica (níspero), Citrus × sinensis (naranjo), Olea europea (olivo), Ficus carica (Higuera), Punica granatum (granado). Estas especies, en su mayoría frutales, han sido escogidas por su alta presencia en nuestro territorio.

a) Ceratonia siliqua

b) Citrus x sinensi

c) Ficus carica

d) Olea siligua

e) Punica granatus

f) Eriobotrya japónica

Ilustración 10: Imágenes de las distintas hojas de las especies seleccionadas

3.3.1. Cálculo de los parámetros identificadores

Todos los objetos que nos rodean pueden describirse físicamente de muchos modos, se pueden tener en cuenta características relacionadas con la forma, el color, la textura, etc. Por tanto, también las hojas, pueden ser clasificadas según una gran variedad de características que las describen. Sin embargo, en esta aplicación, no podemos tener en cuenta todos estos descriptores ya que no podemos prever las condiciones en las que se van a tomar las imágenes, su calidad, iluminación, etc. Por este motivo, vamos a centrarnos sólo en aquellas características que se puedan extraer de la forma de éstas.

Existen bastantes descriptores morfológicos que definen un objeto, los más destacados son:

Descriptores de forma	Descripción
Área	Área del objeto, o número de píxeles que lo forman.
Longitud del Perímetro	Perímetro del objeto o número de píxeles que forman su contorno.
Rectángulo mínimo de cierre	Rectángulo de mínima área posible que encierra el objeto.
Relación de aspecto	Relación longitud/anchura del rectángulo mínimo de cierre.
Elongación	Relación entre el área dividida por el cuadrado de su anchura.
Agujeros	Número de agujeros y sus áreas, que contiene el objeto.
Rectangularidad	Cuanto más se acerque a 1 más cuadrado será el rectángulo mínimo de cierre.
Circularidad	Cuanto más se acerque a 1 más se parecerá a un círculo su elipse.
Momentos	Familia de parámetros que describen la geometría de una región.

Tabla 2: Tipos de descriptores morfológicos

Al tratarse de una aplicación móvil, donde todos los aspectos principales de la imagen son variables, ya que no podemos controlar la iluminación, ni la aproximación del objetivo a la imagen, ni la rotación de la cámara, no todos los factores anteriores son válidos. Tan solo podremos optar por aquellos que sean invariantes con la distancia y con la orientación. Por tanto, se han seleccionado los tres descriptores morfológicos que se han considerado más importantes y representativos: relación de aspecto, compacidad y los momentos invariantes de Hu.

A. Relación de Aspecto

Cómo ya se ha descrito en la tabla anterior, la relación de aspecto es la relación entre la longitud máxima Dmax y la longitud mínima Dmin del mínimo rectángulo delimitador (MBR) o, cómo se ha calculado para esta aplicación, de la elipse equivalente a la figura del objeto. Se ha optado por la segunda opción puesto que para ésta no se requiere conocer la orientación del objeto previamente.

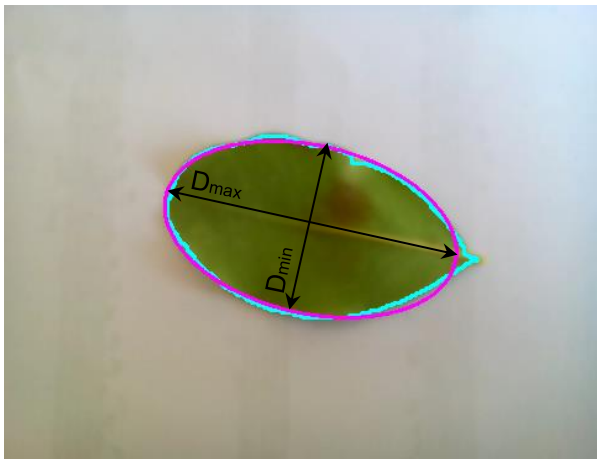


Ilustración 11: Método de cálculo de la relación de aspecto de una hoja

$$AR = \frac{D_{min}}{D_{max}}$$

```
//FitElllipse:
RotatedRect minElpse = new RotatedRect();;
MatOfPoint2f contoursMatofPoints2f = new
MatOfPoint2f(contours.get(poscicionConourMax).toArray());
minElpse=fitEllipse(contoursMatofPoints2f);
//Core.ellipse(src, minElpse, new Scalar(255, 0, 255), 2, 8);
mostrarFoto(src);

Point[] xyz = new Point[4];
minElpse.points(xyz);

//Relacion aspecto
double relacionAspecto;
double heidht = minElpse.size.height;
double width = minElpse.size.width;
if (heidht < width) {
    relacionAspecto = heidht / width;
}else{
    relacionAspecto = width  / heidht;
}
```

B. Compacidad

La compacidad o compactación es un descriptor morfológico que muestra la relación entre el perímetro y el área del objeto. Este se acerca a uno a medida que el objeto se aproxima a un círculo y es menor para objetos elongados.

$$C = \frac{4\pi A}{P^2}$$

Se pueden producir ambigüedades con objetos con geometría no regular, pudiéndose obtener el mismo valor de compacidad para dos objetos de formas diferentes. Sin

embargo esta ecuación es adimensional y por tanto, invariante frente a cambios de escala y frente a rotaciones, lo que la hace un buen descriptor para este proyecto.

C. Momentos invariantes de Hu

La geometría de una región plana se basa en el tamaño, la posición, la orientación y la forma. Todas estas medidas están relacionadas con una familia de parámetros llamada momentos. Los momentos de orden p, q de una imagen $I(x, y)$ se definen como

$$Mpq = \sum_{x=1}^{N} \sum_{y=1}^{M} x^p y^q I(x, y)$$

Por ejemplo, el momento de orden cero para una imagen binaria $M_{00} = \sum_{x=1}^{N} \sum_{y=1}^{M} I(x, y)$ representa el área del objeto.

Para poder hacer una descripción independiente de la posición del objeto, los momentos pueden calcularse respecto a su centro de gravedad. La posición puede ser definida a través de los momentos, tal y como se muestra en la siguiente expresión:

$$\bar{x} = \frac{M_{10}}{M_{00}}, \qquad \bar{y} = \frac{M_{01}}{M_{00}}$$

A éstos se les denomina momentos centrales, se les puede considerar invariantes en cuanto a la posición del objeto y se representan como

$$\mu_{pq} = \sum_{x} \sum_{y} (x - \bar{x})^p (y - \bar{y})^q f(x, y)$$

Al igual que el centro de gravedad, los momentos centrales también se pueden expresar en función de los momentos originales como:

$\mu_{00} = M_{00},$

$\mu_{01} = 0,$

$\mu_{10} = 0,$

$\mu_{11} = M_{11} - \bar{x}M_{01} = M_{11} - \bar{y}M_{10},$

$\mu_{20} = M_{20} - \bar{y}M_{01},$

$\mu_{02} = M_{02} - \bar{y}M_{01},$

$\mu_{21} = M_{21} - 2\bar{x}M_{11} - \bar{y}M_{20} + 2\bar{x}^2 M_{01},$

$\mu_{12} = M_{12} - 2\bar{y}M_{11} - \bar{x}M_{02} + 2\bar{y}^2 M_{10},$

$\mu_{30} = M_{30} - 3\bar{x}M_{20} + 2\bar{x}^2 M_{10},$

$\mu_{03} = M_{03} - 3\bar{y}M_{02} + 2\bar{y}^2 M_{01},$

Debido a que μ_{00} hace referencia al área del objeto, podemos normalizar los momentos para tener una descripción independiente del tamaño, obteniendo así los momentos centrales normalizados.

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{(1+\frac{p+q}{2})}}$$

Finalmente, a partir de los momentos centrales normalizados de orden dos y tres es posible extraer siete parámetros denominados **momentos invariantes de Hu**, independientes a posición, tamaño y rotación del objeto.

$h_1 = \eta_{20} + \eta_{02}$,

$h_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$,

$h_3 = (\eta_{30} - \eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$,

$h_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{03} + \eta_{21})^2$,

$h_5 = (\eta_{30} - 3\eta_{12})(\eta_{03} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{03} + \eta_{21})^2] + (3\eta_{21} - \eta_{03})(\eta_{03} + \eta_{21})[3(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2]$,

$h_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{03} + \eta_{21})$,

$h_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{03} + \eta_{21})^2] - (\eta_{30} - 3\eta_{12})(\eta_{03} + \eta_{21})[3(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2]$.

Estos momentos, serán los que se utilizarán como descriptores morfológicos de las hojas fotografiadas. Cada uno de ellos representa una característica de la forma de los objetos, y OpenCV aporta una serie de funciones que nos facilitan el cálculo.

```
private void momentosHu (final MatOfPoint contorno, final Mat binSrc){
    // Calculamos primero los momentos M_pq a partir del contorno
    Moments momentos = Imgproc.moments(contorno, false);
    Mat hu = new Mat(7,0,Core.DEPTH_MASK_8U);
    // Momentos de HU:
    Imgproc.HuMoments(momentos, hu);

// Extraemos el valor de cada momento de Hu y los guardamos en una
//variable
    double[] hum1 = hu.get(0, 0);
    momentosdeHu[0]=hum1[0];
    Log.d(ArbolApp.TAG, "LabActivty.momentosHu() : h1 =  " + hum1[0]);
}
```

3.3.2. Sistema clasificador

Al final del proceso anterior cada hoja queda definida por un modelo, $D = [d_1, d_2, d_3 \dots d_n]$, compuesto por una serie de primitivas d_n, que corresponden a sus características de forma que sea única y se pueda identificar sin confusión. Para almacenar estos datos de cada hoja, se ha definido un nuevo objeto en la aplicación, denominado `Hoja()`. Además, para este objeto se ha desarrollado el método `comparaHojas()`, que compara dos objetos Hoja y nos devuelve un valor. Este método se verá más detalladamente a continuación.

```
Hoja(double h1, double h2, double h3, double h4, double h5, double h6,
double h7, double relacionAspecto, double compacidad, String nombre);
```

Por tanto, antes de llevar a cabo este apartado, se ha tenido que repetir todo el proceso anterior con distintas hojas de todas las especies elegidas, y con fotografías tomadas con diferentes orientaciones para lograr una base de datos suficientemente amplia como para poder abordar todas las variaciones posibles en las hojas que se fotografíen a posteriori.

Este proceso, se ha realizado con un mínimo de 3 hojas diferentes por cada especie de árbol, y se han intentado realizar al menos dos fotografías diferentes por cada una de ellas. Se debe asegurar que la detección haya sido lo más correcta posible, para evitar en la medida de lo posible falsos resultados. Todos los datos recogidos durante esta pequeña investigación se han ido almacenando en tablas como esta:

Olivo

Momentos	Imagen 1	Imagen 2	Imagen 3	Imagen 4	Imagen 5	Imagen 6
h1	0,506748	0,495325	0,569418	0,429431	0,457672	0,540664
h2	0,229442	0,218175	0,294987	0,157324	0,181417	0,264555
h3	0,008862	0,006752	0,009918	0,005061	0,008446	0,006622
h4	0,007589	0,005735	0,005725	0,004028	0,006516	0,004488
h5	0,000062	0,000036	0,000043	0,000018	0,000048	0,000024
h6	0,003629	0,002676	0,002655	0,001586	0,002700	0,002130
h7	0,000000	0,000000	0,000006	0,000000	0,000001	-0,000002
R/A	0,159137	0,156387	0,136523	0,191459	0,178733	0,149099
Compacidad	0,258069	0,241048	0,208971	0,308454	0,269370	0,256720

Tabla 3: Valores de los descriptores extraidos las distintas imágenes tomadas para una misma especie

La clasificación de la hoja puede realizarse de dos modos diferentes, mediante el algoritmo del vecino más próximo o mediante la comparación con prototipos. El primer caso, consistiría en comparar los nuevos datos, adquiridos de la hoja fotografiada, con todos los datos recogidos en nuestras bases de datos. Este proceso sería demasiado lento y requeriría de más información necesaria en la aplicación, y por tanto se haría mucho más pesada cuanto más ampliáramos el número de especies.

El segundo método, consiste en calcular las medias de cada parámetro, para cada hoja y crear así, un prototipo medio con el que comparar los datos de la nueva

adquisición. Este método agiliza mucho el proceso de clasificación y reduce el tamaño de la aplicación, no aumentando en exceso en caso de ampliar el número de especies a comparar.

Por tanto, las tablas se han ampliado para obtener los mejores resultados posibles. De esta forma, se han calculado la media y la mediana de cada uno de los descriptores. También se han calculado la desviación típica y el coeficiente de variabilidad, de forma que podamos observar que parámetros son más estables y podamos aplicar un sistema de pesos que priorice los descriptores más robustos.

Momentos	Media	Mediana	Desviación	Coeficiente de Variación
h1	0,499876	0,501036	0,051597	0,103220
h2	0,224317	0,223808	0,051002	0,227364
h3	0,007610	0,007599	0,001778	0,233581
h4	0,005680	0,005730	0,001304	0,229498
h5	0,000039	0,000039	0,000016	0,417208
h6	0,002563	0,002666	0,000681	0,265749
h7	0,000001	0,000000	0,000003	3,197984
R/A	0,161890	0,157762	0,020015	0,123633
Compacidad	0,257106	0,257395	0,510554	0,129481

Tabla 4: Valores de la media, mediana, desviación típica y coeficiente de variación de cada descriptor para una única especie

Para calcular los pesos que se aplicarán a cada descriptor y asegurarnos que siempre sea el mismo, se han calculado las medias de cada coeficiente de variación, obteniendo así el coeficiente de variación medio, $\overline{Cv}$, cuya inversa, nos dará el peso de cada descriptor.

En esta tabla, podemos ver perfectamente el grado de robustez que tiene cada uno de los descriptores calculados. Observamos como los primeros momentos de hu y los descriptores de relación de aspecto y compacidad, otorgan una mayor estabilidad. Los descriptores menos robustos, de alguno de los cuales se podría incluso prescindir, tendrán un valor muy poco relevante en la toma de decisión final.

Momentos	Olivo	Algarrobo	Higuera	Naranjo	Granado	Níspero	$\overline{Cv}$
h1	0,1032	0,0090	0,0115	0,0656	0,1138	0,1438	**0,0745**
h2	0,2274	0,0946	0,5122	0,3188	0,6003	0,4251	**0,3631**
h3	0,2336	0,5323	0,1772	0,7584	1,5148	0,5628	**0,6298**
h4	0,2295	0,6985	0,4154	1,4029	1,7004	0,7366	**0,8639**
h5	0,4172	1,2299	0,6059	3,1506	2,2295	1,1093	**1,4571**
h6	0,2657	0,8356	0,4220	2,0256	1,7648	0,8646	**1,0297**
h7	3,1980	0,8437	7,9454	2,8287	2,3702	7,0730	**4,0432**
R/A	0,1236	0,0467	0,0353	0,1112	0,2088	0,1489	**0,1124**
Compacidad	0,1274	0,0271	0,0957	0,1149	0,1673	0,1247	**0,1095**

Tabla 5: Esta tabla muestra los coeficientes de variación de cada descriptor para todas las especies evaluadas y el coeficiente de variación medio calculado

Una vez almacenados todos los datos y calculados los prototipos de cada especie, ya podemos clasificar la hoja según se asemeje más a un prototipo o a otro. Para ello se realiza un simple algoritmo, descrito en el método de la clase Hoja antes nombrado. Este algoritmo consiste en restar, uno a uno, los descriptores de las dos hojas a comparar, multiplicando el resultado de cada suma por la inversa del $\overline{Cv}$. Finalmente el error correspondiente se calcula sumando los resultados de estas operaciones.

$$Error = \frac{\left|d_1 - d_1^{(0)}\right|}{\overline{Cv}_1} + \frac{\left|d_2 - d_2^{(0)}\right|}{\overline{Cv}_2} + \frac{\left|d_3 - d_3^{(0)}\right|}{\overline{Cv}_3} + \cdots + \frac{\left|d_N - d_N^{(0)}\right|}{\overline{Cv}_N}$$

Este proceso se repetirá para cada prototipo, y se compararán los errores resultantes de cada hoja, de forma que, aquel que de un error menor, será el correspondiente a la especie de la hoja fotografiada.

4. RESULTADOS OBTENIDOS

Para realizar una buena evaluación del funcionamiento de esta aplicación se deberían realizar pruebas en contextos muy variados de iluminación, orientación de las hojas, grado de perpendicularidad con respecto al plano fotografiado, color del fondo, etc. Se han intentado llevar a cabo multitud de pruebas, con tal de esclarecer el buen funcionamiento de la aplicación.

El paso más delicado de todo el proceso, es la correcta detección de la hoja, por ello es donde más esfuerzo se ha realizado. Aunque un buen reconocimiento no significa que la clasificación vaya a ser correcta, es muy probable que ésto ocurra así, mientras que en la mayoría de casos, una mala detección sí suele significar una mala clasificación.

Hemos comprobado que la mayoría de hojas bien detectadas suelen dar buenos resultados en la clasificación final, sin embargo, existen problemas en la clasificación de la hoja del granado, puesto que tiene una morfología bastante variable, entre hojas de un mismo árbol. Por esta razón, el porcentaje de aciertos se ha reducido bastante para este tipo de hojas, que muy a menudo la aplicación confunde con la del naranjo.

Ilustración 12: Fotografías de distintas hojas del mismo árbol de la especie granado

También se debe prestar mucha atención a las diferencias entre hojas de distintos árboles de una misma especie. Estas pueden variar en tamaño forma y color según la época del año o según el lugar donde se encuentre, y por tanto, una buena base de datos es muy importante. Se puede plantear el uso de varios prototipos de una misma especie para aquellos casos donde las diferencias sean muy claras y elevadas.

Se ha observado que las hojas fotografiadas al revés, es decir, la parte trasera hacia delante, dan un mejor resultado en la segmentación de la imagen, ya que esta zona de la hoja suele ser más mate y por tanto reduce los brillos y destellos de la hoja.

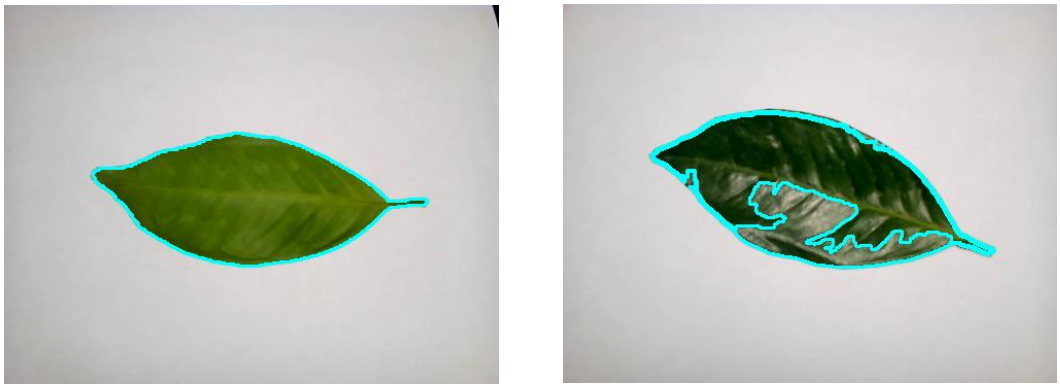


Ilustración 13: Misma hoja fotografiada por ambas caras en las mismas condiciones de iluminación. Podemos observar la selección del contorno en cada caso.

También se ha comprobado que una hoja previamente aplanada, colocando peso sobre ella, favorece la detección debido a que se reduce considerablemente la cantidad de sombra proyectada sobre el fondo.

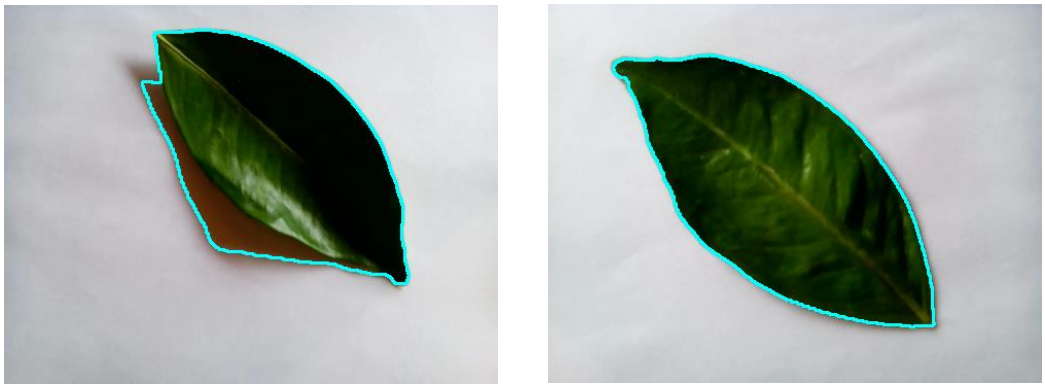


Ilustración 14: Misma hoja fotografiada antes (a la derecha) y después (izquierda) de ser aplanada

Hemos realizado pruebas con fondos de diferente color. Se ha fotografiado una misma hoja, bajo las mismas condiciones de luz con diferentes fondos para esclarecer cuáles de ellos dan un mejor resultado en la detección. De este pequeño estudio se ha podido concluir que la aplicación funciona en mayor medida con colores claros y poco saturados y mejora si estos son neutros, marrones o azules.

Ilustración 15: Imágenes tomadas de una misma hoja, bajo las mismas condiciones de iluminación, pero con fondos de diferentes colores. Podemos observar la detección realizada para cada caso.

5. APLICACIONES Y MEJORAS FUTURAS

Como ya se ha explicado, durante la realización de este proyecto se ha desarrollado un prototipo de una aplicación móvil, y por tanto, el número de mejoras en este programa puede ser bastante amplio. Para una versión futura de la aplicación se han planteado una serie de mejoras que no se han realizado en este primer prototipo por falta de tiempo o porque no han sido consideradas prioritarias.

- Se plantea la posibilidad de mejorar las bases de datos mediante una **red neuronal de aprendizaje** o algún otro algoritmo de *Machine Learning*. De este modo, cada vez que el usuario realice una detección satisfactoria, tanto los datos recopilados, como la fotografía realizada puedan ser enviados a un servidor, mejorando así, el prototipo calculado de cada especie almacenada.

- **Ampliar el número de especies**. Esta es una operación bastante sencilla, pero se debe tener en cuenta que cuanto mayor sea el número de éstas, mayor será, también, la posibilidad de error. Por tanto, sería conveniente, plantear varias soluciones para cada nueva hoja fotografiada y ordenarlas según su parecido, siempre y cuando no superen un rango determinado. Esta operación puede ir acompañada de varias imágenes, de forma que el usuario pueda identificar visualmente otras características, que le aporten la información necesaria para decidir de qué especie se trata.

- Otro avance importante podría ser la realización de **la segmentación y elección del contorno a tiempo real**. De esta forma, el usuario, podría observar antes de tomar la fotografía, como se está produciendo la segmentación, pudiendo, así, tomar decisiones previas como el cambio de posición para evitar reflejos y/o sombras, mejorar la iluminación de la escena, etc.

- La introducción de **nuevos descriptores** morfológicos e incluso, podría estudiarse la posibilidad de incluir algún descriptor de color.

- **Mejorar la interface gráfica**. Como ya se ha explicado en el apartado 3.1. el peso de este proyecto ha recaído sobre el desarrollo de un algoritmo eficiente de reconocimiento y clasificación de hojas, por lo que se ha dejado en segundo lugar el diseño de una interface gráfica adecuada. Esto es muy importante, tratándose de este tipo de productos, que pretenden llegar siempre a la mayor cantidad de gente posible, por eso, el diseño de una buena interface, que sea intuitiva y práctica es fundamental para la comercialización de una aplicación como ésta.

- Se podría valorar el hecho de poner algún objeto que nos sirva de referencia dentro de la imagen, cómo podría ser una moneda o una regla milimetrada, a fin de poder obtener el tamaño real de la hoja basándonos en algo conocido.

Hoy por hoy, el prototipo desarrollado para este trabajo, no tiene una aplicación real, ya que el número de especies evaluadas es muy reducido. Sin embargo, una correcta ampliación de éste, podría generar una serie de salidas comerciales. Quizás sería muy pretencioso intentar desarrollar una aplicación capaz de clasificar cualquier árbol de la península, sin embargo, puede ser interesante introducirla en parques naturales o zonas con un número de especies más limitado, cómo granjas escuelas, aulas de educación ambiental o parques y jardines.

6. CONCLUSIONES

El reconocimiento de imágenes es una disciplina que siempre ha causado gran interés entre la sociedad científica, siendo cada vez más precisa y compleja. Existen muchas técnicas para desarrollar sistemas capaces de reconocer imágenes, como en cualquier campo, el problema reside en la elección de la técnica adecuada según la funcionalidad final que se desea aportar a la aplicación.

Durante el desarrollo de este trabajo se ha programado una aplicación móvil capaz de reconocer y clasificar hojas de árboles a través del reconocimiento de imágenes tomadas con nuestro dispositivo y se ha logrado desarrollar un algoritmo fiable y eficiente. Sin embargo se es consciente de que, tanto el algoritmo como la aplicación, pueden mejorarse en muchos aspectos, y que para ello se debe continuar trabajando.

Por tanto, se ha considerado que los resultados obtenidos son adecuados para un trabajo de esta envergadura. Tratándose de una aplicación móvil donde los parámetros principales de iluminación, orientación y perpendicularidad son imposibles de controlar, el porcentaje de aciertos difícilmente pueda ser muy elevado.

Se debe considerar también que la biblioteca utilizada para desarrollar el algoritmo de reconocimiento de imágenes, de OpenCV, está diseñado para trabajar correctamente en el lenguaje de programación C++. Sin embargo, esta aplicación ha sido desarrollada por completo en Java, y por tanto, aunque este lenguaje es soportado por OpenCV, nos hemos encontrado con muchos problemas a la hora de encontrar información o tutoriales que explicaran el correcto funcionamiento e implementación de las funciones que nos ofrece esta biblioteca.

Finalmente, cabe destacar que el esfuerzo realizado durante la realización de este trabajo, ha sido plenamente satisfactorio, puesto que se ha conseguido desarrollar una aplicación real, que pese a ser un simple prototipo, puede llegar a tener gran proyección si se continúa trabajando en él. Este trabajo ha favorecido el crecimiento y mejora a nivel personal, y ha aportado una serie de habilidades y conocimientos nuevos que sin duda ayudarán en la vida profesional.

7. BIBLIOGRAFÍA

[1] Joseph Howse, *"Android Application Programming with OpenCV: Build Android apps to capture, manipulate, and track objects in 2D and 3D"* - Published by Packt Publishing Ltd. ISBN 978-1-84969-520-6, 2011.

[2] Samarth Brahmbhatt, *"Practical OpenCV"* - Copyright © 2013, Distributed to the book trade worldwide by Springer Science+Business Media New York, ISBN-13 (electronic): 978-1-4302-6080-6.

[3] Kenneth Dawson-Howe, *"A practical introduction to computer vision with OpenCV"* - © 2014 John Wiley & Sons Ltd, Registered office by John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom.

[4] David Knight, James Painter, Matthew Potter, *"Automatic Plant Leaf Classification for a Mobile Field Guide. An Android Application"* - Stanford University Department of Electrical Engineering, California.

[5] Neeraj Kumar, Peter N. Belhumeur, Arijit Biswas, David W. Jacobs, W. John Kress, Ida Lopez, and Joâo V. B. Soares, *"Leafsnap: A Computer Vision System for Automatic Plant Species Identication"* - University of Washington, Seattle WA, Columbia University, New York NY, University of Maryland, College Park MD, National Museum of Natural History, Smithsonian Institution, Washington DC

[6] Web de los desarrolladores del sistema operativo para smartphones y tablets Android: https://developer.android.com/sdk/index.html

[7] Web oficial de la biblioteca de visión artificial OpenCV: http://opencv.org

[8] Página web de preguntas y respuestas sobre questiones de programación general: http://stackoverflow.com

[9] Web de la aplicación para dispositivos móviles Pl@ntNet: http://www.plantnet-project.org/papyrus.php?langue=en

[10] Arbolapp: http://www.arbolapp.es/

[11] Comparativa entre los softwares Android Studio y Eclipse: http://academiaandroid.com/android-studio-v1-caracteristicas-comparativa-eclipse/