UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

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

Desarrollo de un software de adquisición, procesado y reconocimiento de textos manuscritos.

TRABAJO FINAL DE GRADO

Autor/a:
Guillermo Roselló Gil

Tutor/a:
José Ignacio Herranz Herruzo

GANDIA, 2016

Resumen

El objetivo de este trabajo es la implementación de algoritmos de reconocimiento de caracteres en conjunto con sistemas de tratamiento de imagen para corregir errores provocados a la hora de capturar la imagen. La meta final consiste en obtener un archivo de texto que pueda ser manipulable a partir de la imagen de un documento impreso o manuscrito.

Palabras clave

OpenCV, Tesseract, Mejora de imagen, Reconocimiento de caracteres, Android.

Abstract

This project aims to the implementation of character recognition algorithms along with image improvement methods in order to correct errors given in the capture process. The final goal is to obtain a editable text file from an image of a printed or hand-written document.

Key words

OpenCV, Tesseract, Image enhancement, Character recognition, Android.

Índice de Contenidos

1. Introducción

De un tiempo a esta parte los dispositivos móviles han ido ganando fuerza y potencia a pasos agigantados, consiguiendo así hacerse un hueco en nuestra vida diaria hasta el punto en el que ya a pocos se les hace posible imaginar el día a día sin su Smartphone.

Todo este potencial almacenado en nuestros bolsillos se puede explotar muy fácilmente y la digitalización de toda la información es algo que ya ha dejado de ser descabellado para convertirse en algo que es cuestión de tiempo que pase. A este objetivo se llegará gracias a los algoritmos de reconocimiento, que simulan el procesado ojo-cerebro que nosotros usamos sin darnos cuenta.

1.1 Estructura del proyecto

Para una mejor compresión de los temas que se van a tratar a lo largo del trabajo, aquí se va a hacer un pequeño resumen de los contenidos y la estructura del proyecto. La memoria comienza con una breve introducción a las tecnologías OCR, su historia y el estado actual de éstas; para luego explicar el funcionamiento del algoritmo de reconocimiento Tesseract, detallando todos los pasos que realiza y como administra cada paso. Por último, se expondrá el funcionamiento de la aplicación creada, explicando paso a paso como trabaja y finalizando con los resultados obtenidos y las conclusiones del proyecto.

1.2 Motivación del proyecto

Este proyecto nace de la unión de dos de mis facetas más personales. Por una parte, los estudios de ingeniería que he estado cursando estos últimos años, más específicamente aquellos conocimientos sobre el tratamiento de imagen; por otra parte, mi afición a las historias de fantasía y ciencia ficción que me llevaron a indagar más y más en este tema hasta el punto de crear mis propias historias.

Un día surgió el problema de no poder compartir estas pequeñas historias por mi afición a escribirlo todo a mano y pensar en pasarlo todo a ordenador era un engorro. Afortunadamente, descubrí que existían ciertos algoritmos dedicados al reconocimiento de caracteres. En ese momento supe que tendría que guiar mi proyecto por ese camino, ya que no solo me ayudaría a mí, sí no que también ayudaría a digitalizar obras en formato físico para que su conocimiento se preservara más tiempo.

2. Optical Character Recognition (OCR)

“Los sistemas que, a partir de un texto escrito o impreso en papel o similar, crean un fichero de texto en un soporte de almacenamiento informático, se denominan sistemas OCR (Optical Character Recognition)” [1].

2.1 Historia

Dos personas fueron las primeras en tener una patente relacionada con la tecnología OCR. El primero fue Gustav Tauschek, que obtuvo su patente en Alemania durante el año 1929; el segundo fue Handel el año 1933 en Estados Unidos. Después Tauschek también consiguió una patente de tecnología OCR en Estados Unidos en el año 1935.

La máquina de Tauschek era un dispositivo mecánico que se basaba en foto-receptores, de manera que cuando el carácter coincidía con el de la plantilla, un rayo de luz era dirigido hacia él.

La tecnología OCR se sume en un sueño profundo durante los siguientes años, seguramente por el bajo nivel de la tecnología, hasta por lo menos el año 1950, cuando se le pregunta a David Shepard, un criptoanalista en la agencia de seguridad de las fuerzas armadas de los Estados Unidos, si era posible que se construyera una máquina que convirtiera mensajes impresos en lenguajes que pudieran ser almacenados en un ordenador. Después de la afirmativa de David, creó la máquina Gismo y en 1953 la mostró al público. Ese mismo año, después de conseguir la patente de Gismo, creó IRM, una empresa donde desarrollaría el primero de varios sistemas OCR.

El primer sistema OCR comercial fue instalado en Reader Digest en el año 1955 y así, el mundo del OCR se empezó a expandir. El servicio postal de Estados Unidos usa tecnología OCR desde el año 1965 y ese mismo año se implantó por primera vez en Europa, en el mundo de la banca. Canadá, por su parte, implementó la tecnología OCR en su sistema postal el año 1971, este sistema se usa para crear un código de barras con toda la información del sobre, tal como el nombre del destinatario y la dirección.

2.2 Estado actual

La tecnología OCR considera actualmente, el reconocimiento exacto de la escritura en base latina, un problema ya solucionado. La precisión de la detección de caracteres excede el 99%, aunque en muchos casos no llega a ser suficiente, requiriendo de supervisión humana para la corrección de los posibles errores. Pero hay que tener en cuenta que este porcentaje solo es aplicable a los textos impresos, cuando se trata de reconocimiento de caracteres en texto manuscrito, se habla de una probabilidad de acierto en el reconocimiento entre el 80 y el 90% por página. Esto nos puede llevar a pensar que sigue siendo un porcentaje alto de acierto, pero teniendo en cuenta la cantidad de caracteres que nos podemos encontrar en un texto, se puede traducir en decenas de errores por página, limitando su uso a contextos muy concretos.

El reconocimiento de texto cursivo es un campo de investigación activo con porcentajes de reconocimiento mucho más bajos que los del reconocimiento de texto manuscrito. Los estudios han mostrado que la única manera de conseguir que este porcentaje aumente es obteniendo información tanto contextual como gramatical, para así reducir el abanico de posibilidades a la hora de reconocer una palabra. Para problemas más complejos se usa reconocimiento inteligente de carácter, debido a que las redes neuronales que los componen trabajan de manera indiferente a las transformaciones lineales o no lineales del proceso que las compone.

- **Ventajas:**
  - Búsqueda y recuperación de documentos
  - Explotación de los documentos
  - Perspectiva económica
- **Inconvenientes:**
  - Carencia de conocimiento y expertos.
  - Elevado coste de generación con todas las funciones
  - Nivel de efectividad insatisfactorio en el reconocimiento de documentos históricos.

2.3 Algoritmos de reconocimiento

Con el desarrollo del potencial de las tecnologías OCR salieron a la luz diferentes empresas dedicadas a crear software relacionado y algunas de ellas han desarrollado sus propios algoritmos de reconocimiento de caracteres.

Se puede encontrar un gran número de algoritmos de reconocimiento., implementados por una gran variedad de software OCR, siendo estos cuatro los más usados:

- **ABBYY finereader:** Este software fue creado por la empresa ABBYY en el año 1991 y lanzado dos años después. Capaz de trabajar con todos los lenguajes existentes e incluso lenguajes de programación. Permite convertir imágenes y archivos PDF, documentos digitales y archivos de imagen a archivos editables, además de soportar el formato Ebook usado en los libros electrónicos. Nos encontramos ante un software de pago, con un precio muy elevado.
- **ReadIris:** Al igual que el programa de ABBY, este es capaz de convertir todo tipo de archivo con letra impresa y manuscrita en un archivo editable. Es capaz de trabajar junto con Microsoft Office y OpenOffice para obtener directamente los archivos en la extensión deseada. También cuenta con la funcionalidad de generar archivos Web-friendly. Desafortunadamente nos encontramos con que este software también es de pago.
- **Omnipage:** Este producto fue creado por Nuance Communications en el año 1980 y causó una gran impresión debido a la poca oferta que se encontraba en esa época. Al igual que los ya mencionados softwares nos encontramos con que es capaz de reconocer caracteres en archivos de imagen y PDFs para añadirlos a archivos de texto modificables, tanto de toda la suite de Microsoft como de Adobe, capaz de manejar hasta 120 lenguajes. También permite acceso a la nube, pero por desgracia también es de pago.
- **Tesseract:** Herramienta de software creada en 1985 y liberada en 2005 que actualmente se encuentra en desarrollo por Google y lo distribuye Apache. Es considerado uno de los motores libres de OCR con una mayor precisión y además tiene la capacidad de poder ser entrenado para que cada vez reconozca mejor un idioma. Este software sí que es libre y es el que ha sido elegido para realizar este proyecto.

2.4 Aplicaciones similares

Gracias al desarrollo de estos algoritmos de reconocimiento de caracteres podemos encontrar infinidad de aplicaciones, tanto de ordenador como de Smartphone, que usan OCR para reconocer textos. La mayoría de las aplicaciones de escritorio son conversores de PDF a archivo de texto, usando los sistemas OCR para ello. Aquí se puede ver una pequeña lista con algunos programas que usan esta tecnología:

- **Free OCR for Windows:** Es una herramienta sencilla con una interfaz gráfica intuitiva donde podemos pasarle tanto un archivo de imagen o un PDF, lo procesa y te da la opción de convertirlo en un archivo Word o copiarlo directamente al portapapeles. Podemos encontrarlo en la página de la tienda de Microsoft tanto para Windows 8.1 como para la versión de Windows 10 y puede reconocer hasta 21 idiomas
- **OCR Terminal:** Es un método online de detección de texto que usa el motor OCR ABBYY. Es una aplicación de pago por lo que no se ha podido probar como funciona.
- **Text Fairy:** Esta es una aplicación para Android, recientemente incluida en el Android Market. A primera vista es muy intuitivo y nos deja seleccionar el tipo de entrada, desde cámara o desde archivo y nos muestra el avance del reconocimiento, pero con la primera prueba de texto escrito a mano el procentaje de éxito no llega ni siquiera al 10% de similitud con el texto original.
- **CamScanner:** Aplicación bastante usada por universitarios para escanear documentos con el móvil. Añade la posibilidad de usar la tecnología OCR, pero con un resultado bastante semejante al de Text Fairy.

3. Tesseract

3.1 Historia

Tesseract empezó como un proyecto de doctorado en los laboratorios de HP en Bristol y rápidamente ganó popularidad como un posible añadido para tecnologías propias de HP. La motivación de este proyecto surgió debido a que las tecnologías OCR de la época estaban en pañales y solo funcionaban con la mejor calidad de impresión, dejando al resto de documentos inservibles para ello. Después del trabajo conjunto de los laboratorios HP de Bristol y la división de escáneres de HP en Colorado, Tesseract se convirtió en el líder de las tecnologías OCR, por encima de las tecnologías comerciales, pero no llegó a convertirse en un producto.

El desarrollo continuó hacia adelante y el estudio se centró más en mejorar la eficacia de rechazo que en la precisión básica. En 1994 acabó este proyecto, y con él, se detuvo el desarrollo de Tesseract. En 1995 se envió a la Universidad de Las Vegas, a la prueba anual de precisión de tecnologías OCR, donde demostró su valía frente a las demás tecnologías OCR comerciales.

3.2 Arquitectura y funcionamiento

Tesseract se basa en un algoritmo de funcionamiento paso por paso. Empieza con un análisis de los componentes para obtener los contornos de los caracteres, y aunque fue una decisión muy cara tanto de diseño como de programación, otorga una ventaja bastante significativa ya que convertía la detección de texto negro sobre fondo blanco en una tarea trivial simplemente analizando el anidado de los contornos.

El siguiente paso, ya con los contornos en nuestro poder, es reunirlos en *blobs*, expresión que viene del inglés para referirse a una masa informe o un amasijo de cosas. Se usa esta expresión debido a que el algoritmo junta contornos cercanos en una misma área para luego procesarlos de manera individual.

A continuación estos blobs se organizan en líneas de texto para evitar confusiones y se asigne una palabra a la línea que no corresponde. Cuando las líneas ya están asignadas se procede entonces a descomponer las líneas en palabras y se hace el primer reconocimiento de texto. Tesseract funciona de manera que persigue el menor número de palabras posibles por lo que intenta reconocer todo el texto palabra por palabra. Aquellas palabras que hayan sido reconocidas satisfactoriamente serán agregadas al entrenador y las que no se convertirán en *fuzzy spaces* para que sean reconocidas después. Por último, se hace una segunda pasada de reconocimiento, ahora con la ayuda del entrenador, para resolver los fuzzy spaces y se extrae el archivo digital con el resultado.

El proceso descrito se podría resumir en los siguientes pasos:

- Análisis y almacenamiento de contornos.
- Los contornos se agrupan en blobs.
- Los blobs se organizan en líneas de texto.
- Las líneas se descomponen en palabras.
- Primer intento de reconocimiento.
- Las palabras satisfactorias son añadidas al entrenador.
- Segundo intento de reconocimiento, ahora con la ayuda del entrenador.
- Resolución de los fuzzy spaces
- Extracción del texto digital.

Este sería el funcionamiento general del algoritmo Tesseract, pero vamos a ver con un poco más de detalle como realiza cada acción.

3.2.1 Line finding

El algoritmo de búsqueda de líneas funciona de tal manera que se pueda reconocer una página que esté inclinada sin necesidad de tener que hacer ninguna modificación y así evitar cualquier pérdida de calidad que se pudiera originar debido a esto. Las partes fundamentales de este proceso son el filtrado de los blobs y la construcción de las líneas.

Una de las ventajas con las que partía el desarrollo de Tesseract es que HP ya había desarrollado su propia tecnología de análisis de posición de hoja y fue añadida al algoritmo, por lo que Tesseract suponía que el input iba a ser un archivo de imagen binaria con regiones poligonales opcionales ya definidas. Asumiendo que el analizador ha funcionado correctamente y nos ha dado regiones de texto de un tamaño más o menos regular, se aplica un *percentile height filter* para separar caracteres muy grandes y caracteres que se tocan verticalmente. Luego, un *median height filter* aproxima el tamaño del texto de la región por lo que podemos eliminar blobs que sean más pequeños. Estos suelen ser signos de puntuación, marcas diacríticas y ruido.

Los blobs filtrados suelen seguir el mismo modelo de líneas que no se superponen, que sean paralelas, pero que están inclinadas. Mediante un procesado y selección a partir de la coordenada X podemos asignar blobs a una sola línea para así evitar que una palabra sea asignada a una línea que no le corresponda en caso de que la página esté inclinada. Por último, se aplica un método matemático de mínimos cuadrados para la estimación de parámetros. Este se usará para hacer una aproximación de cuál es la línea de la base de las líneas de texto, y así poder añadir los signos de puntuación y las marcas diacríticas.

3.2.2 Baseline fitting

Una vez se han identificado las líneas de texto, las líneas de referencia se ajustan con mayor precisión usando un *Quadratic Spline*, una función de interpolación matemática. Esta parte del proceso es la clave que ayuda a Tesseract a procesar las páginas que están inclinadas, que suelen ser la mayoría de ellas.

Las líneas de referencia se ajustan dividiendo en grupos los blobs dependiendo de lo alejados estén de lo que sería la línea recta. A continuación se aplica el Quadratic Spline al grupo más poblado de blobs ya que se supone que es el grupo más cercano a la línea de referencia. El problema es que el Quadratic Spline puede fallar cuando se necesitan muchos segmentos de spline, en ese caso se aconseja más usar un Cubic Spline.

3.2.3 Fixed Pitch Detection and Chopping

En este punto Tesseract ya ha detectado las líneas y las ha ajustado para que estén dispuestas de manera correcta, por lo que ahora se va a centrar en diferenciar las palabras entre ellas y, a ser posible, descomponer estas palabras en caracteres. Lo primero que va a hacer el algoritmo es separar las palabras en dos grupos, el primero será donde Tesseract incluirá las palabras con caracteres monoespaciados, es decir, aquellas palabras que sus caracteres ocupen el mismo espacio; el segundo grupo estará formado por aquellas palabras que no están formados por caracteres monoespaciados.

Los caracteres monoespaciados permiten a Tesseract descomponer estas palabras en caracteres con un simple filtro de distancias en el eje X, como se puede apreciar en la Figura 1. Estas palabras no serán pasadas ni por el separador ni por el asociador, ya que ya están descompuestas a su mínimo nivel.

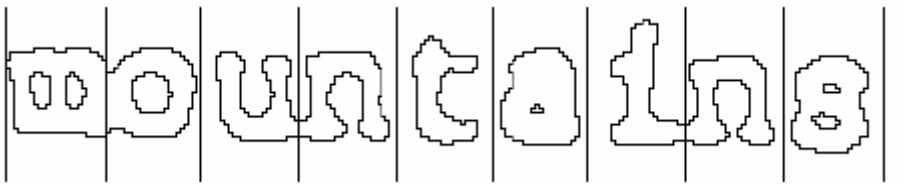

Figura 1 separación horizontal de caracteres

3.2.4 Proportional Word Finding

El segundo grupo, el que contiene el resto de palabras no monoespaciadas, se les llama *proportional text*. La tarea de detectar este tipo de palabras es todo lo contrario a trivial, como podemos apreciar en la figura 2:

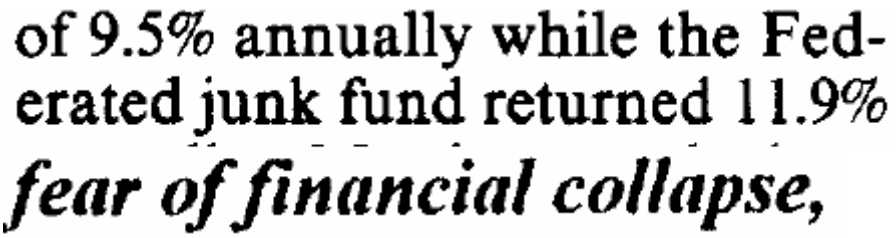

Figura 2 Ejemplo de proportional text

Se puede ver que el espacio que separa “11.9%” del resto es más grande que el que separa “erated” de “junk”. Tesseract resuelve esto mediante la medición de los huecos en un rango vertical entre la línea de base y la línea de la media. Los huecos que no puedan ser diferenciados entre espacio intercaracter o espacio interpalabra se convertirán en espacios Fuzzy y serán resueltos en la segunda pasada del algoritmo.

3.2.5 Word recognition

Parte del proceso de cualquier motor de reconocimiento de caracteres es identificar como una palabra tiene que ser descompuesta en caracteres. Como ya se ha visto en el apartado anterior, las palabras con caracteres monoespaciados se pueden descomponer fácilmente en caracteres, pero aquellas que no lo son requieren de un grado de procesado mayor.

Lo primero que hará Tesseract será coger el Blob menos probable de ser reconocido y lo troceará. Los puntos de corte elegidos serán aquellos puntos cóncavos obtenidos después de aplicar una aproximación poligonal a los contornos. Se necesitan por lo menos tres puntos de corte para poder extraer un carácter de la tabla ASCII satisfactoriamente.

Los cortes se van aplicando por prioridad y cualquier corte que no ayude a mejorar el resultado de la predicción serán deshechos, pero no eliminados en caso de que el asociador lo necesite luego.

Cuando todas las posibles combinaciones de cortes han sido aplicadas y no se ha podido obtener los caracteres se le pasa al asociador. El asociador hace una búsqueda A*, un algoritmo de reconocimiento de caminos, usando todas las combinaciones de cortes que se hayan hecho.

3.2.6 Classifier

Se ha llegado a un punto en el que todas las palabras han sido descompuestas en caracteres y ahora se tiene que decidir que caracteres son, para poder finalizar el proceso de reconocimiento. Tesseract inspecciona las características topológicas de estos caracteres, aunque no sirven las características más destacables, porque es muy fácil encontrar a alguien que escriba una letra de manera diferente a como se suele hacer y siga siendo totalmente válida. Tesseract extrae de los contornos las características más pequeñas y que sean de tamaño fijo, para superponerlas al modelo extraído de los archivos de entrenamiento.

Con todo esto en su poder, Tesseract procede a clasificar los caracteres. Primero se crea una lista de posibilidades con las que podría coincidir el carácter desconocido, después, de cada una de las posibilidades se obtiene un prototipo y se procede a comparar cada uno de estos prototipos con el carácter desconocido. El proceso de comparación consiste en comparar distancias entre el carácter desconocido y el posible prototipo, aquel que nos dé menores distancias será el más probable a que sea nuestro carácter.

3.2.7 Resultados esperados

Podemos ver en la Tabla 1 cuales fueron los resultados del test que pasó en la universidad de Nevada y una comparación del mismo test con una versión más avanzada de Tesseract.

		Character			Word		
Ver	Set	Errs	%Err	%Chg	Errs	%Err	%Chg
HP	bus	5959	1.86		1293	4.27	
2.0	bus	6449	2.02	8.22	1295	4.28	0.15
HP	doe	36349	2.48		7042	5.13	
2.0	doe	29921	2.04	-17.68	6791	4.95	-3.56
HP	mag	15043	2.26		3379	5.01	
2.0	mag	14814	2.22	-1.52	3133	4.64	-7.28
HP	news	6432	1.31		1502	3.06	
2.0	news	7935	1.61	23.36	1284	2.62	-14.51
2.0	total	59119		-7.31	12503		-5.39

Tabla 1 Tabla resultados del algoritmo Tesseract.

En la Tabla 1 se pueden ver los resultados de la prueba realizada en la universidad de Nevada, marcada como HP, y la realizada posteriormente con un Tesseract más avanzado, nombrada como 2.0. En la tabla podemos ver las cantidades de errores por página tanto en carácter como por palabra. Se observa curiosamente que los resultados de la versión 2.0 han salido peor que los de la Universidad de Nevada. Este hecho se puede deber a que las condiciones de la prueba de 1995, en cuanto a calidad de imagen, fueran mejores que las de la segunda prueba.

4. Desarrollo del proyecto

En este apartado se va a detallar el material que se ha necesitado a la hora de desarrollar este proyecto. Se nombrarán todos ellos, tanto software como hardware y se explicará por qué se ha elegido cada uno de ellos.

4.1 Hardware utilizado.

Se ha utilizado un ordenador que cumple los requisitos que pide Android Studio, los cuales son:

- Sistema operativo Windows (10/8.1/7/8/7/Vista preferiblemente 64 bits)
- Mínimo 2GB de memoria RAM, 4GB recomendados.
- 400 MB mínimos de espació libre en el disco duro.
- Una pantalla de resolución mínima 1200x800.
- Tener instalado Java Development Kit (JDK) 7.
- Se requiere al menos 1GB de memoria para Android SDK, emulador de imágenes del sistema y caché.

Para probar la aplicación se ha usado como Smartphone un terminal Android, en este caso un Motorola Moto G XT1320, con la Versión 5.1 de Android instalada. El principal requerimiento de este dispositivo Android era que tuviera cámara y alguna aplicación que permita explorar los archivos del teléfono.

4.2 Software utilizado

Para el desarrollo de este proyecto se necesitaban tres cosas fundamentales: un *framework* donde poder crear la aplicación de Android, una librería que nos permitiera trabajar con la imagen y el algoritmo de reconocimiento de caracteres.

4.2.1 Android Studio

Android Studio es un entorno de desarrollo integrado para la plataforma Android, desarrollado por Google y publicado en 2014. Está basado en el software Intellij IDEA de JetBrains y es publicado de forma gratuita a través de la licencia Apache 2.0. Está disponible tanto para Windows, Mac y Linux.

A su salida desbancó a Eclipse como IDE oficial para el desarrollo de aplicaciones para Android, pero la migración Eclipse-Android Studio es lenta.

Se puede descargar gratuitamente desde la plataforma de Android [Web] y su instalación y configuración es muy sencilla.

Las principales características que incluye Android Studio son:

- Soporte para desarrollo de aplicaciones para Android Wear.
- Herramienta Lint (Detecta código no compatible entre arquitecturas diferentes o código confuso que no es capaz de controlar el compilador) para detectar problemas en 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 Stuido, utiliza Ant.

- Posibilita el control de versiones accediendo a un repositorio desde el que poder descargar Mercurial, 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 decisión de seleccionar este IDE para desarrollar la aplicación se basa en que ya se disponía de conocimientos y experiencia anteriores, así como el hecho de que Eclipse va camino de la obsolescencia.

4.2.2 Biblioteca OpenCV

OpenCV es un conjunto de Bibliotecas de código abierto bajo la licencia BSD (licencia que permite el uso de código fuente en software no libre), desarrolladas por Intel y disponibles desde 1999. La biblioteca está escrita en C y C++ y se puede ejecutar desde diversos sistemas operativos como GNU/Linux, Windows y Mac.

OpenCV fue diseñado ofreciendo un código muy eficiente y con un fuerte enfoque a aplicaciones capaces de ejecutarse en tiempo real y para ello toma ventaja de los procesadores multinúcleo. 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.

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

Se puede recortar imágenes, mejorarlas mediante 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, además del uso de cámaras estéreo para obtener una visión 3D del mundo, y muchas funciones más. Los módulos que componen esta librería están altamente optimizados para funcionar en tiempo real de manera muy eficaz.

De todos los módulos de los que dispone OpenCV solamente se han necesitado tres de ellos: Core, Imgproc e Imgcodecs.

El módulo Core es el que se ocupa de gestionar las estructuras de datos, tipos de datos y gestión de la memoria. El módulo Imgproc se ocupa del filtrado de imágenes, transformaciones geométricas de imágenes, estructura y análisis de formas. Por último, el módulo Imgcodecs es el que se ocupa de leer y escribir imágenes en la memoria.

Aun así, los módulos no son lo único que tiene. OpenCV incluye una clase para poder representar imágenes, la clase Mat. Un objeto de tipo Mat consta de dos partes:

- Cabecera: Conjunto de metadatos que permiten manejar las imágenes. Estos datos pueden incluir:
    - Anchura y altura en pixeles de la imagen.
    - Información sobre el tipo de dato de los pixeles.
    - Información sobre el número de canales de la imagen.

    - Puntero a los pixeles.
- Pixeles: Zona de memoria donde físicamente reside el valor de los pixeles.

Además, cuenta con la instrucción *Submat()* que permite procesar solo un trozo de una imagen grande. De esta manera se puede limitar el procesado a partes específicas de la imagen y, por tanto, mejorar la velocidad de procesado.

También podemos encontrar los siguientes aspectos básicos con los que podemos trabajar si usamos la clase Mat:

- Averiguar el número de canales de una imagen.
- Extraer un canal de una imagen.
- Descomponer una imagen de color en sus componentes.
- Cambiar el formato de una imagen.
- Cambiar una imagen de un tipo de dato a otro.

De este modo podemos acceder a casi cualquier característica de la imagen muy fácilmente. Cuando se haya terminado de trabajar con ella se puede guardar fácilmente como cualquier archivo de imagen.

4.2.3 Tess-two

Tess-two es una librería que incluye el algoritmo Tesseract, también llamado Tesseract Android Tool. Consiste en un conjunto de APIs y archivos necesarios para las librerías del algoritmo de reconocimiento de caracteres. También está disponible para la librería de procesado de imágenes Leptonica, que constituye a una alternativa muy válida a OpenCV.

Tess-two únicamente contiene tres módulos:

- Tess-two: Contiene las herramientas para la compilación de las librerías Tesseract y Leptonica para su uso en la plataforma Android.
- Eyes-two: Contienen código para el procesado adicional de imágenes. Este código se ha copiado de Eyes-free Project y no es necesario para el funcionamiento del módulo Tess-two.
- Tess-two-test: Contiene herramientas para el testeo del módulo Tess-two y, al igual que Eyes-two, no es necesario para el funcionamiento de Tess-two.

Para el caso de esta aplicación solamente nos interesa el módulo Tess-two, donde se encuentra el algoritmo que va a ser usado para reconocer los caracteres de la imagen procesada.

El inconveniente que trae esta librería es que viene sin preparar, por lo tanto, si se importa directamente a nuestro proyecto en Android no funcionará. Para poder conseguir que se importe como una librería Android vamos a necesitar cierto software.

El principal problema viene dado a la hora de que el algoritmo Tesseract está escrito en lenguaje C/C++ y la aplicación Android está programada en Java. Para poder traducir los archivos de la librería y que no haya problemas de compatibilidad se necesita el conjunto de herramientas Android NDK. Estas herramientas permiten implementar dentro de aplicaciones Android trozos de código programado en lenguaje C/C++. De esta manera podemos usar la librería Tes-Two dentro de nuestra aplicación, sin que haya problemas de compatibilidad. Una vez traducida la librería de C/C++ a Java se tendrá que actualizar el SDK de nuestro proyecto usando simplemente la línea de

comando *Android update Project –target 1 [path del proyecto]*. El último paso para poder usar la librería en Android Studio es crear una *build* de esta librería por lo que vamos a necesitar el software de Apache Ant, ejecutando la instrucción *ant release* para crear el build. Hecho esto ya se puede importar la librería y funciona perfectamente.

5. Funcionamiento de la aplicación

Una vez detalladas las herramientas empleadas y el funcionamiento del algoritmo OCR Tesseract, ha llegado el momento de hablar sobre el funcionamiento de la aplicación que se ha desarrollado para este proyecto.

El algoritmo que se ha seguido a la hora de implementar esta aplicación es el proceso estándar de reconocimiento de caracteres, que se puede apreciar en la Figura 3.

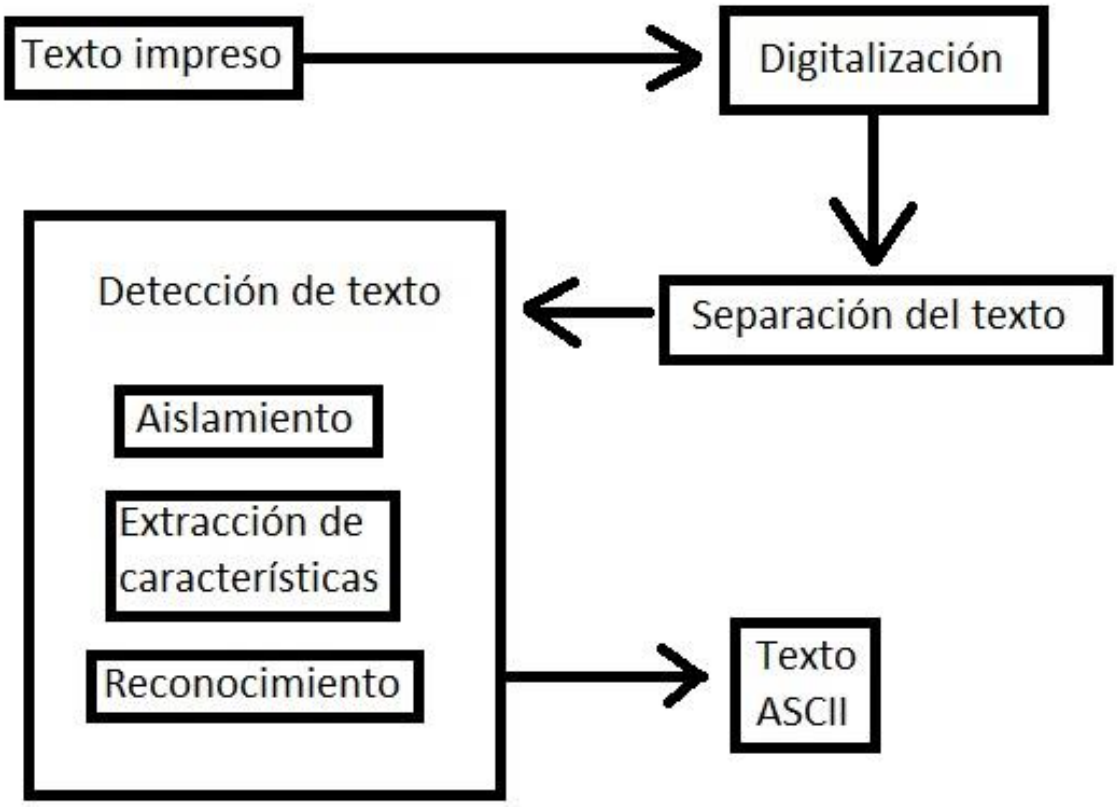

Ilustración 3 Algoritmo OCR

Se pueden observar tres grupos diferenciados en este algoritmo. El primero es la digitalización o captura del texto a reconocer, la separación de los caracteres del resto de la imagen y por último el grupo de reconocimiento del texto. En este apartado se va a explicar cómo se han implementado estas distintas funciones. Hay que remarcar que en este proyecto no se ha dedicado mucho tiempo al diseño de la aplicación, ya que se ha considerado fuera del alcance del objetivo y de las competencias propias. Los esfuerzos se han centrado en la implementación del código relacionado con la captura, mejora y reconocimiento del texto.

Entrando más en detalle en los bloques del algoritmo de reconocimiento encontramos:

- **Texto impreso:** Esta fase se refiere exclusivamente al texto que se va a usar, al que va a ser reconocido. Como se ha mencionado anteriormente, tiene que ser texto manuscrito o impreso, pero no puede ser cursivo, ya que eso atañe a otra competencia.
- **Digitalización:** En esta fase se procederá a realizar la captura digital de la imagen del texto impreso mediante el uso de una cámara fotográfica, en este caso, con la cámara que viene en el terminal Android.

- **Separación del texto:** Las competencias de este bloque son las de tratar la imagen con el propósito de corregir errores y preparar la imagen para ayudar al algoritmo a la hora del reconocimiento. Aquí es donde se realiza la ecualización del histograma, la umbralización del texto y la apertura morfológica.
- **Detección del texto:** En este punto es cuando entra en juego el algoritmo Tesseract, ya que este bloque es el encargado de reconocer el texto. Se puede ver que este bloque está subdividido en otros tres, tal y como se ha explicado en el apartado anterior.
- **Texto ASCII:** Una vez procesado el texto de manera satisfactoria, obtenemos un archivo de texto con la información que ha sido extraída de la imagen.

Una de las características a la hora del desarrollo de una aplicación Android es que se pueden separar en clases. De esta manera se puede dividir el código en diferentes "ventanas", evitando así sobrecargar el MainActivity con demasiado código y ayudando a la organización de este y a su eventual depuración.

Los componentes principales que se pueden añadir a una aplicación Android son:

- **Activity:** Representa 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 cualquiera otro lenguaje visual.
- **View:** Son los componentes básicos con los que se construye la interfaz gráfica de la aplicación, similar a los controles de Java o .NET. Se tiene la posibilidad de usar todos los controles básicos que Android da, o se pueden crear unos propios.
- **Service:** 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 acción, por ejemplo actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales.
- **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, estructura, o implementación.
- **Broadcast Receiver:** es un componente destinado a detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema o por otras aplicaciones.
- **Widget:** Los widgets son elementos visuales, normalmente interactivos, que pueden mostrarse en la pantalla principal (*home screen*) del dispositivo Android y recibir actualizaciones periódicas. Permite 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, inciar otra aplicación, etc.

La aplicación desarrollada está dividida en tres clases: MainActivity, donde se captura y almacena la imagen tomada por la cámara; Procesador, donde se realiza el

procesado de la imagen; y, por último, Tess, donde se llama al algoritmo de reconocimiento.

5.1 Captura y almacenamiento

En este apartado se va a hablar sobre la captura de la imagen que posteriormente será procesada y reconocida. Este se considera un aspecto importante ya que sobre una buena base los siguientes pasos podrán ser más eficientes.

En esta clase es donde se gestiona la cámara, inicialización, resoluciones, captura y almacenamiento. Por decisiones de diseño se ha decidido que la aplicación solo trabaje con fotos tomadas de la cámara.

Al iniciar la aplicación podemos ver directamente la imagen que está tomando la cámara, se ve en blanco y negro por decisión de diseño, ya que eso luego ayuda a la hora del procesado. Podemos observar el resultado en esta imagen.

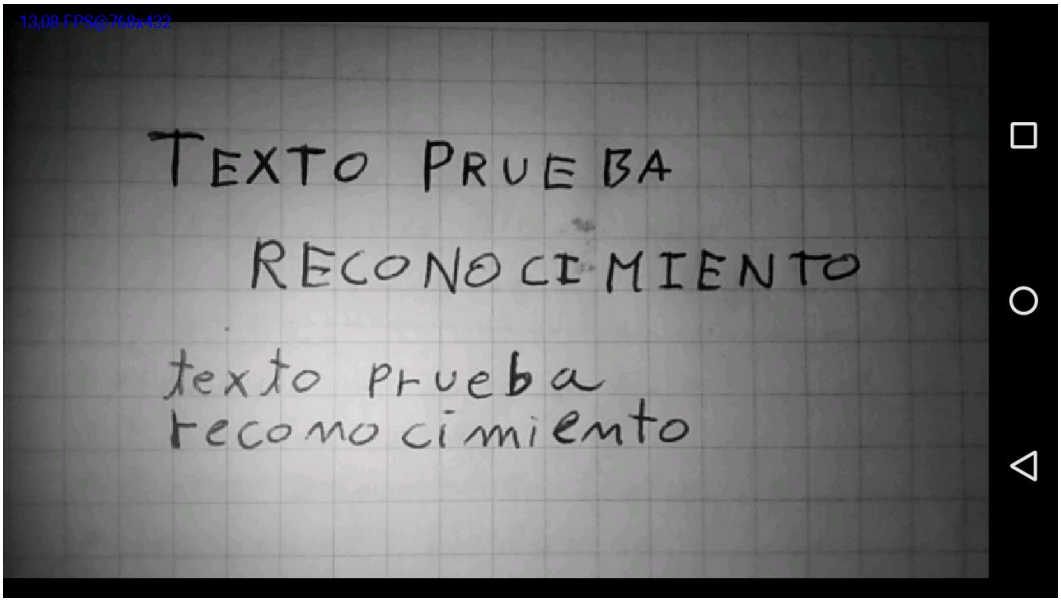

Figura 4 Captura Pantalla

Para conseguir que la imagen se vea en escala de grises, y así aligerar el proceso de ecualización posterior,se ha usado la siguiente línea de código:

```
Imgproc.cvtColor(output,mBgr, Imgproc.COLOR_GRAY2BGR, 3)
```

Estando en esta pantalla, si se hace un toque a la pantalla del dispositivo Android se despliega un menú.

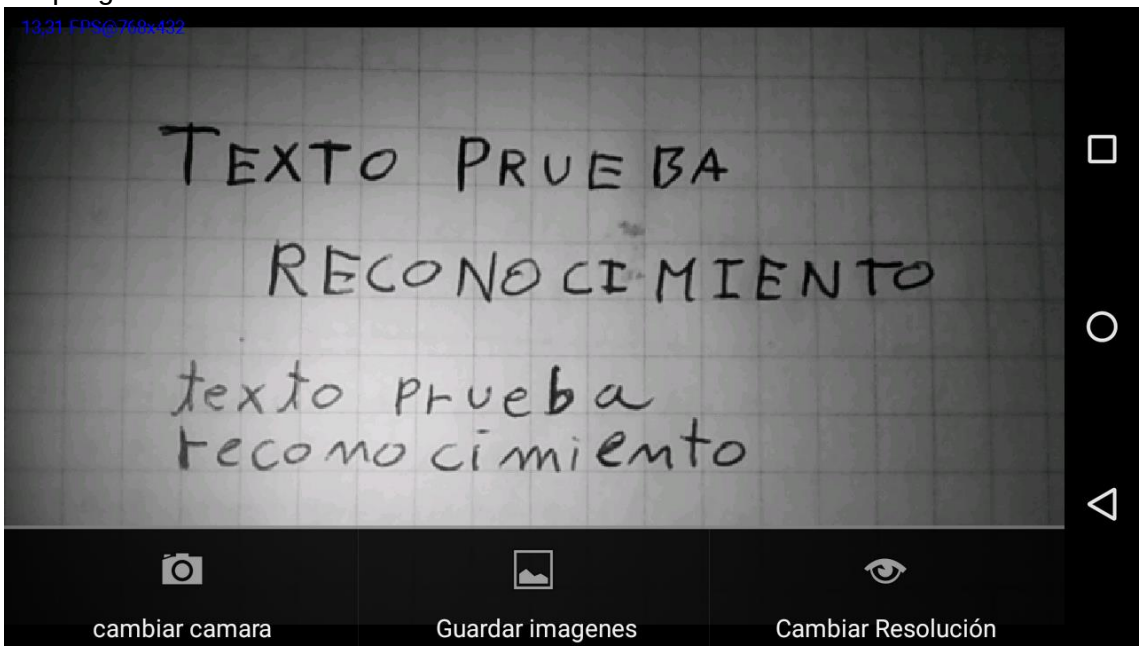

Figura 5 Menú aplicación

Se puede ver que aparecen tres opciones:

- **Cambiar cámara:** Permite alternar entre la cámara frontal y la trasera.
- **Guardar imagen:** Permite almacenar en memoria una captura del frame que se encuentre en pantalla en ese mismo momento.
- **Cambiar resolución:** Permite elegir entre una de las tres resoluciones disponibles.

Esta aplicación está diseñada para que abra la cámara y no requiera de terceras aplicaciones para gestionarla. De esta manera se tiene pleno acceso a la imagen y se puede trabajar con ella directamente a través de OpenCV.

La mayoría del código que encontramos en la clase MainActivity se centra en gestionar la cámara y el almacenamiento, así que solo se va a mostrar el código más relevante.

```
public void onCameraViewStarted(int width, int height){
    cam_altura = height;
    cam_anchura = width;
    procesador = new Procesador();
    tess = new Tess();
```

Lo primero que hace la aplicación al iniciarse es ajustar la resolución predefinida e inicializar las llamadas a las otras clases. La resolución predefinida de esta aplicación es de 800x600. En caso de que se use un terminal Android que no llegue a tal resolución, se ajustará automáticamente a la resolución máxima permitida.

```
public Mat onCameraFrame(CvCameraViewFrame inputFrame){
    Mat entrada = inputFrame.gray();
    Mat salida = procesador.procesa(entrada);
```

```
    if (guardarSiguienteImagen){
        takePhoto(entrada,salida);
        guardarSiguienteImagen = false;
    }
    if (tipoEntrada > 0)
        Imgproc.resize(salida,salida,new Size(cam_anchura,cam_altura));
    return salida;
```

En el código de arriba se puede ver la clase OnCameraFrame, donde se realiza el procesado de cada frame y donde se llamará a la función TakePhoto, encargada de almacenar la captura en memoria cuando se pulse el botón “Guardar Imagen”.

El código relacionado con la captura del frame es el siguiente:

```
private void takePhoto(final Mat input, final Mat output){
    //Determina la ruta para crear los archivos
    final long currentTimeMillis = System.currentTimeMillis();
    final String appName = getString(R.string.app_name);
    final String galleryPath =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES).toStrin
g();
    final String albumPath = galleryPath + "/"+appName;
    final String photoPathIn = albumPath+"/In_"+currentTimeMillis+".bmp";
    final String photoPathOut = albumPath+"Out_"+currentTimeMillis+".png";
    MatOfInt matOfInt= new MatOfInt();
    matOfInt.fromArray(Imgcodecs.CV_IMWRITE_PXM_BINARY,1);

    //Hay que asegurarse de que el directorio existe
    File album = new File(albumPath);
    if (!album.isDirectory() && !album.mkdirs()){
        Log.e(TAG, "Error al crear el directorio"+ albumPath);
        return;
    }
    //intenta crear los archivos
    Mat mBgr = new Mat();
    if (output.channels() == 1)

        Imgproc.cvtColor(output,mBgr, Imgproc.COLOR_GRAY2BGR, 3);
    else
        Imgproc.cvtColor(output,mBgr,Imgproc.COLOR_RGBA2BGR,3);
    if (!Imgcodecs.imwrite(photoPathOut, mBgr)){
        Log.e(TAG, "Fallo al guardar "+ photoPathOut);
    }
    if (input.channels() == 1)
        Imgproc.cvtColor(output, mBgr, Imgproc.COLOR_GRAY2BGR, 3);
    else
        Imgproc.cvtColor(output, mBgr, Imgproc.COLOR_RGBA2BGR, 3);
    if (!Imgcodecs.imwrite(photoPathIn, mBgr, matOfInt)){
        Log.e(TAG, "Fallo al guardar "+ photoPathIn);
    }
    File file = new File(photoPathIn);
    tess.Detectar(file, albumPath);
    mBgr.release();


    return;
```

Lo primero que se hace es determinar la ruta donde se va a almacenar dicha imagen. Se ha decidido crear una carpeta, con el mismo nombre que la aplicación, en la raíz del

dispositivo Android. El nombre del archivo será el tiempo en milisegundos en el que se ha tomado la foto. Para almacenar la imagen se ha usado el comando Imgproc.imwrite(), al cual hay que pasarle tres argumentos. El primero es un String con la dirección de la carpeta donde se almacenará el archivo, el segundo es la imagen en formato Mat que se quiere guardar y, por último, un MatOfInt donde se guardan las características ya predefinidas para darle formato a la imagen. En este caso se ha elegido un formato .bmp ya que es el formato que pide Tesseract. MatOfInt es una clase que hereda de la clase Mat y es, básicamente, un array de Ints que puede trabajar con el formato Mat.

5.2 Procesador

Como se ha mencionado en el apartado anterior, en la función OnCameraFrame se llama a la clase que se ocupa de procesar los frames que le van entrando. El código relativo a esta parte se encuentra en la clase Procesador.

A la hora de reconocer un texto en una imagen digital, el preprocesado de la imagen es una parte muy importante ya que es donde se corrigen los posibles defectos que se hayan dado a la hora de tomar la foto y se prepara para el reconocimiento.

Dependiendo del tipo de dispositivo Android que se use la calidad de la cámara varía, por ello lo primero que se realiza en el preprocesado es un filtro paso bajo de tipo blur. Los filtros paso bajo se ocupan de eliminar las frecuencias altas de la imagen, de esta manera se consigue suavizar los contrastes de la imagen y así eliminar gran parte del ruido que se pueda meter en la imagen.

Para conseguir este objetivo se crea una máscara de pixeles con la forma de una matriz y cada uno de los componentes estará dividido por el número de casillas que se encuentren en la matriz, es decir, si tenemos una matriz de 3x3 el resultado será el siguiente:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Esta mascara se aplicará a la imagen que se desee tratar. Usando el centro de la máscara como referencia, se le hará pasar por todos los pixeles, se multiplicará ese valor, tanto en el central como en los pixeles vecinos, y se usará el valor de la suma de los resultados como nuevo valor del pixel.

Para obtener este efecto se ha usado la siguiente línea de código.

```
Imgproc.blur(Input, Output, Kernel Size);
```

Los argumentos que hay que incluir en esta función para que funcione son sencillos, solo hay que indicarle los Mat de entrada y destino e indicarle el tamaño de la máscara que se va a aplicar sobre la imagen.

En la figura 6 se puede ver el efecto de este filtro sobre la imagen original mostrada en la figura 4.

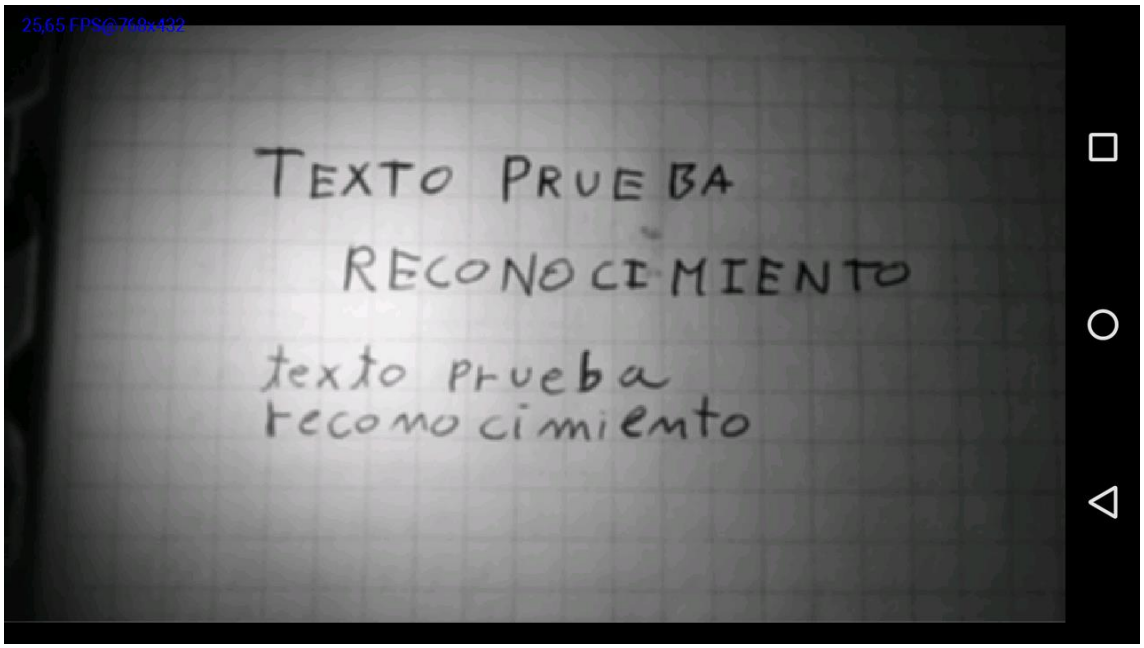

Figura 6 Filtro blur

Se ha decidido usar un filtro no muy agresivo ya que no se quiere distorsionar demasiado los caracteres para no dificultar el proceso del reconocimiento, para ello se ha elegido una matriz pequeña de 3x3.

Una vez se ha aplicado el blur se procede al siguiente paso del preprocesado, la ecualización del histograma con el fin de mejorar el contraste de la imagen. Para ello se usa el siguiente comando:

```
Imgproc.equalizeHist(input,output);
```

Los argumentos que se han usado son el Mat de entrada que se obtiene de la salida del blur, para poder continuar con la cadena de procesado de imagen, y un Mat vacío donde se guardará la imagen resultante. Este proceso lo hace de manera automática, sin tener que modificar la curva, extrae el histograma y lo normaliza para ajustar los niveles de grises. Se puede observar el resultado de la ecualización se puede ver en la figura 7.

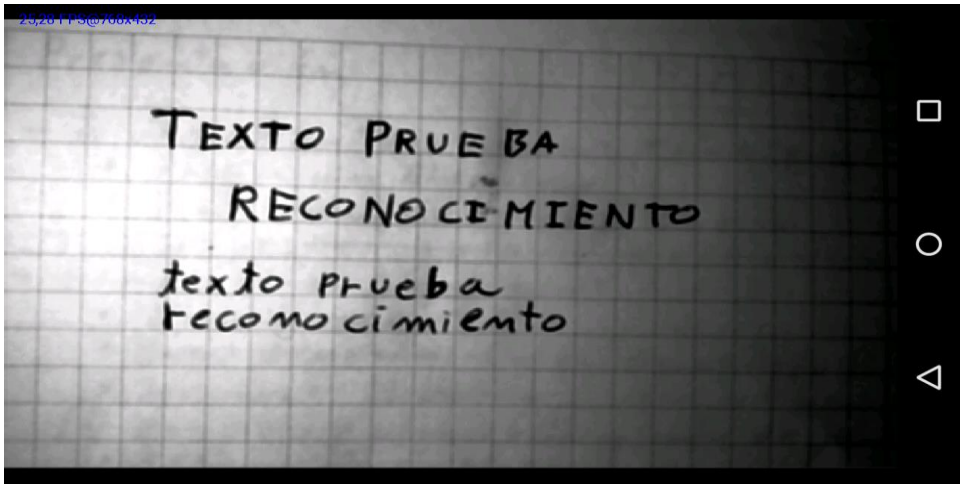

Figure 7 Ecualización del histograma

El siguiente paso del preprocesado consta de una umbralización adaptativa, para ello se ha usado el siguiente código:

```
Imgproc.adaptiveThreshold(input,output,maxValue,adaptativeMethod,thresholdType,BlockSize,
C);
```

Esta función se usa para separar entre las letras y el fondo blanco que se supone que es el papel, optimizando así el posterior proceso de reconocimiento. Se ha elegido una umbralización adaptativa debido a que los resultados de una umbralización predeterminada no se ajusta a los niveles de iluminación cambiantes a lo largo del papel. Lo que hace la función adaptativeThreshold() es calcular un umbral diferente para cada punto de la imagen y de esta manera ajustando el umbral a cada zona de la imagen.

Los argumentos usados han sido los siguientes:

- **Input:** Imagen a umbralizar, que será el resultado de la ecualización.
- **Output:** Destino de la imagen umbralizada.
- **maxValue:** Nivel que asigna la umbralización. Se ha decidido poner el valor máximo ya que se busca el alto contraste letra/papel.
- **adaptativeMethod:** Aquí se indica que tipo de algoritmo se usa. En este caso se ha usado ADAPTATIVE_TREHS_MEAN_C debido a que el valor del umbral se establece mediante la media general de los vecinos.
- **thresholdType:** Este argumento indica si la umbralización se hará de manera normal o inversa, se ha elegido normal por la misma razón que el valor de maxValue.
- **BlockSize:** Determina el tamaño del bloque que se usará para decidir los vecinos.
- **C:** Valor que indica cuanto tiene que desviarse el valor del pixel de la media para ser umbralizado.

Para los valores de BlockSize y C se han escogido los valores 15 y 5, correspondientemente, debido a que son los que mejor se ajustan al tamaño de los caracteres con los que se trabaja. Se puede observar el resultado en la figura 8.

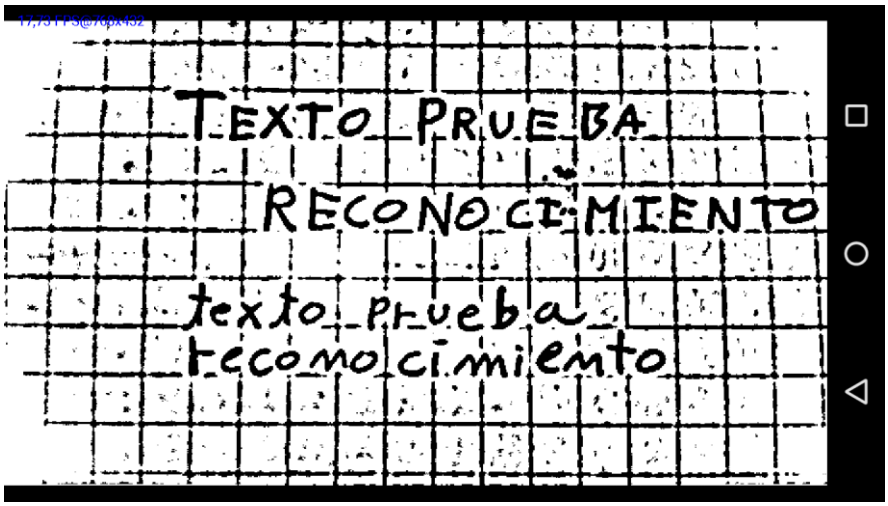

Figure 8 Umbralización

Una vez realizada la umbralización se observa como los caracteres se han podido separar del fondo, pero también se detectan zonas de ruido que Tesseract puede confundir con caracteres, por lo que es necesario una limpieza de ruido. Para realizar este cometido se ha usado el operador morfológico de apertura, usando las siguientes líneas de código:

```
Imgproc.erode(salida, salida,
Imgproc.getStructuringElement(Imgproc.CV_SHAPE_RECT, new Size(2,2)));
Imgproc.dilate(salida,salida,Imgproc.getStructuringElement(Imgproc.CV_
SHAPE_RECT,new Size(2,2)));
```

La función de apertura morfológica consiste en aplicar dos operadores morfológicos elementales, conocidos como erosión y dilatación, para realzar las formas de los objetos. Estos operadores elementales funcionan de la siguiente manera.

La primera función que se aplica es erode. Esta es una función de degradación, pues elimina los pixeles que sean más pequeños que el elemento geométrico determinado, en este caso un rectángulo de 2x2 pixeles, tamaño elegido debido a que era el máximo posible sin que se deterioraran demasiado los caracteres. Esto debería eliminar la mayoría de errores ya que se cuentan como pequeños puntos negros aleatorios.

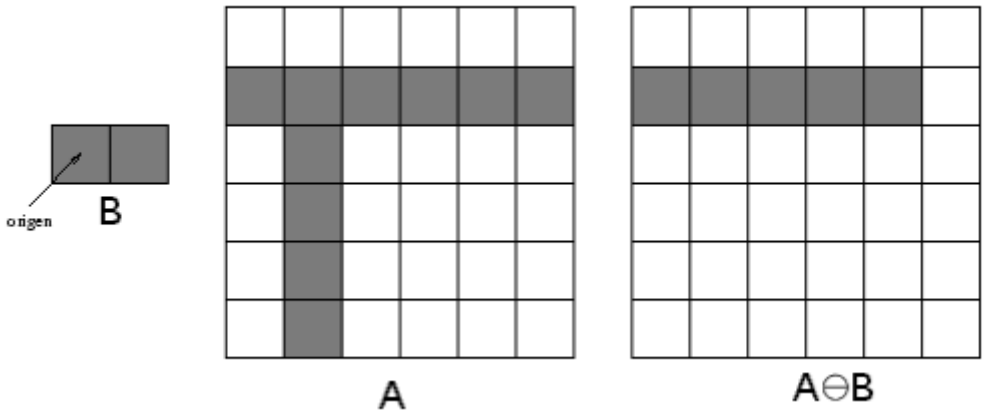

Figura 9 Ejemplo erode

La figura 9 es un claro ejemplo de cómo funciona un erode. Tenemos la imagen a modificar que está etiquetada como A y por otra parte tenemos el elemento geométrico que va a ser usado como máscara, etiquetado como B. Erode funciona de la siguiente manera, hace pasar el centro de la máscara por cada uno de los pixeles y elimina aquellos pixeles que sean, en conjunto, más pequeños que la máscara, por eso se observa que la fila de pixeles verticales se elimina después de realizar el erode.

La siguiente función aplicada es dilate, la función hermana de erode, que corrige los caracteres defectuosos que ha dejado erode. Dilate funciona prácticamente igual que erode, la única diferencia es que al contrario de eliminar pixeles, los crea.

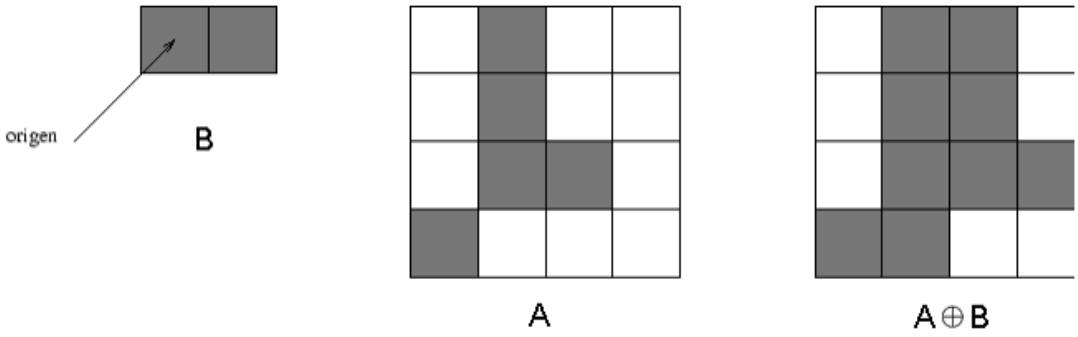

Figura 10 Ejemplo Dilate

En la figura 10 se puede ver un ejemplo de esta función. Se encuentran los mismos elementos, la imagen a tratar y la máscara, pero esta cuando llega a un pixel con información rellena los alrededores que se encuentren vacíos.

Con la suma de estas dos funciones se pueden eliminar errores de digitalización más pequeños que los caracteres manteniendo estos relativamente intactos.

Al finalizar el proceso de apertura morfológica se obtiene la imagen en la figura 11.

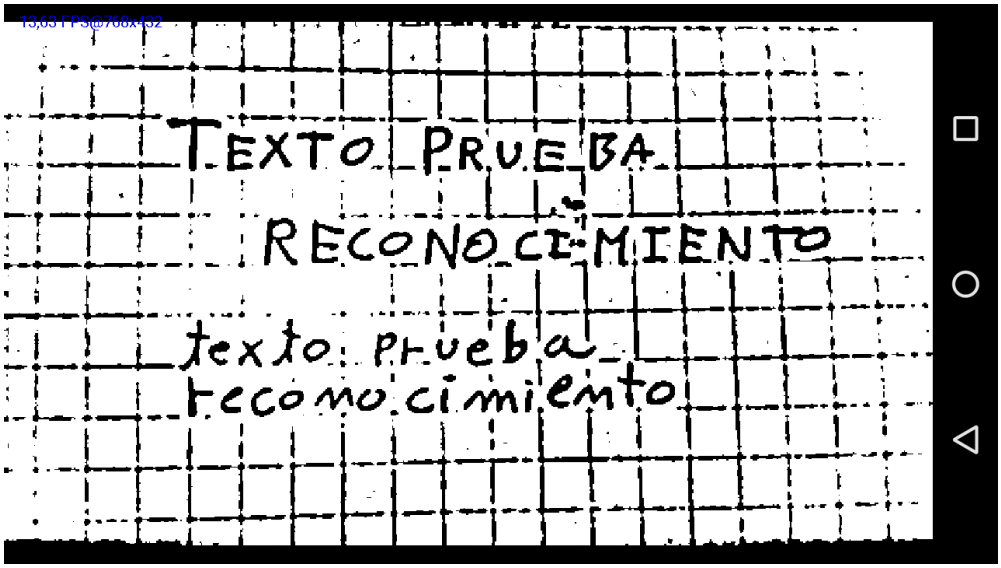

Figure 11 open

Se puede observar como la mayor parte del ruido incluido en la umbralización se ha eliminado. Esta será la imagen que se almacenará en memoria

5.3 Reconocimiento

En este punto del algoritmo, el texto ha sido digitalizado y tratado, por lo que solo queda la fase de reconocimiento. La llamada a la clase que se ocupa de contener el código de reconocimiento es la siguiente:

```
File file = new File(photoPathIn);
tess.Detectar(file);
```

Con la primera línea se crea una instancia de la función File para que se cree un archivo con la imagen. Luego esa imagen se le pasa a la función Detectar() de la clase tess, donde se ha introducido el código relativo al algoritmo. El código dentro de la función detectar es el siguiente:

```
public void Detectar (File file)  {
    String langpath = "/mnt/sdcard/tesseract-ocr-master/";
    TessBaseAPI tessBaseAPI = new TessBaseAPI();
    tessBaseAPI.init(langpath, "spa");
    tessBaseAPI.setImage(file);
   String extext = tessBaseAPI.getUTF8Text();
    File Extext = new File("/mnt/sdcard/pdfoutput/Deteccion");
    try {
        FileWriter Writer = new FileWriter(Extext);
        Writer.append(extext);
        Writer.flush();
        Writer.close();
    }catch (IOException e){
        System.out.println("Fallo creación txt");
    }
    tessBaseAPI.end();
```

Primero se llama al constructor del algoritmo con la función TessBaseAPI y para poder inicializarlo se recurre a la línea tessBaseAPI.Init(). Los argumentos que necesita el algoritmo para inicializarse son dos, el primero es la dirección de la carpeta donde se encuentran los datos del entrenador del algoritmo, previamente copiados al dispositivo Android; el segundo es para indicarle a Tesseract que idioma se va a usar. Ahora se le tiene que indicar a Tesseract que archivo tiene que reconocer, para ello se usa la línea tessBase API.setImage() que admite diferentes tipos de input, pudiendo ser imágenes en bitmap, archivos o imágenes en formato pixa, que es el que usa la librería Leptonica. En este caso se ha usado la opción del archivo ya que OpenCV trabaja con el formato Mat, que es irreconocible por Tesseract, por lo que primero se ha almacenado y ahora se le pasa la dirección donde se ha guardado dicha imagen.

Ahora existen dos posibilidades a la hora de extraer el texto. Por una parte se puede usar la clase TessPDFRenderer y crear un archivo PDF con toda la información extraída, o se puede extraer en un String y luego incluirlo en un archivo de texto. Por motivos de diseño y de proximidad al objetivo de este proyecto se ha decidido extraer el texto mediante la línea de código tessBaseAPI.getUTF8Text() en un String y luego escribir dicho texto en un archivo de texto con formato .txt.

6. Resultados.

Como se ha hablado anteriormente en este trabajo, la precisión del algoritmo Tesseract no es del 100%, por lo que se espera una cierta cantidad de errores en el resultado. Además de que el resultado también se verá influido por factores externos al algoritmo, como por ejemplo, la resolución y la calidad de la cámara, la iluminación, la calidad del papel y el tipo de texto que se va a reconocer, manuscrito o impreso.

Para probar esta aplicación se ha mantenido siempre la misma resolución de 800x600 y se ha probado con diferentes textos de los dos tipos de letra que se pretende reconocer, manteniendo tanto la iluminación como el lenguaje a reconocer en todos los pasos.

La primera prueba se ha realizado con papel cuadriculado y texto manuscrito.

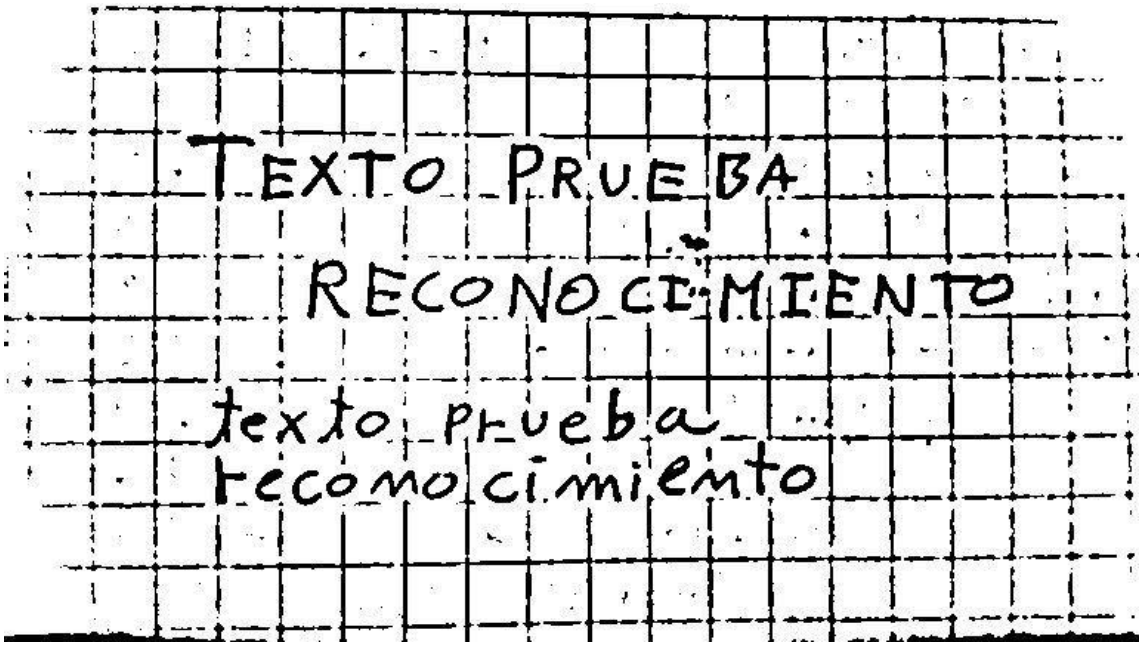

Figura 12 Primera prueba

-L! ï- ' —IIIIIIII_1;_;_J
'JÏLT mi P III II'SJM
¿4 ' ' "EXTQ- _R u.' BA i L.
4 ,;_¿_¿¿_3_LLEJFJ_'E ' .Q-..'
i *' r-._ .qu_chI< N- «1
:Ïg» ¡4-!¿2MÏl Tal u
¿J 'rCXJïQá—PhüábïefiéJr" ' , H
'-.Ï' 1 .Cca_Mo cum} M.,0 ¿"44'

Fïïü rHfiw

| mi: ___' J 4- ._ "L ._.._.._.

Figura 13 Resultado primera prueba

Se puede observar que la tasa de reconocimiento para este caso ha sido práctimente del 0%. Esto se debe a que la cuadricula que hay detrás dificulta el preprocesado a la hora de eliminar ruido y también confunde a Tesseract.

La segunda prueba se ha realizado esta vez con papel en blanco, usando el mismo texto.

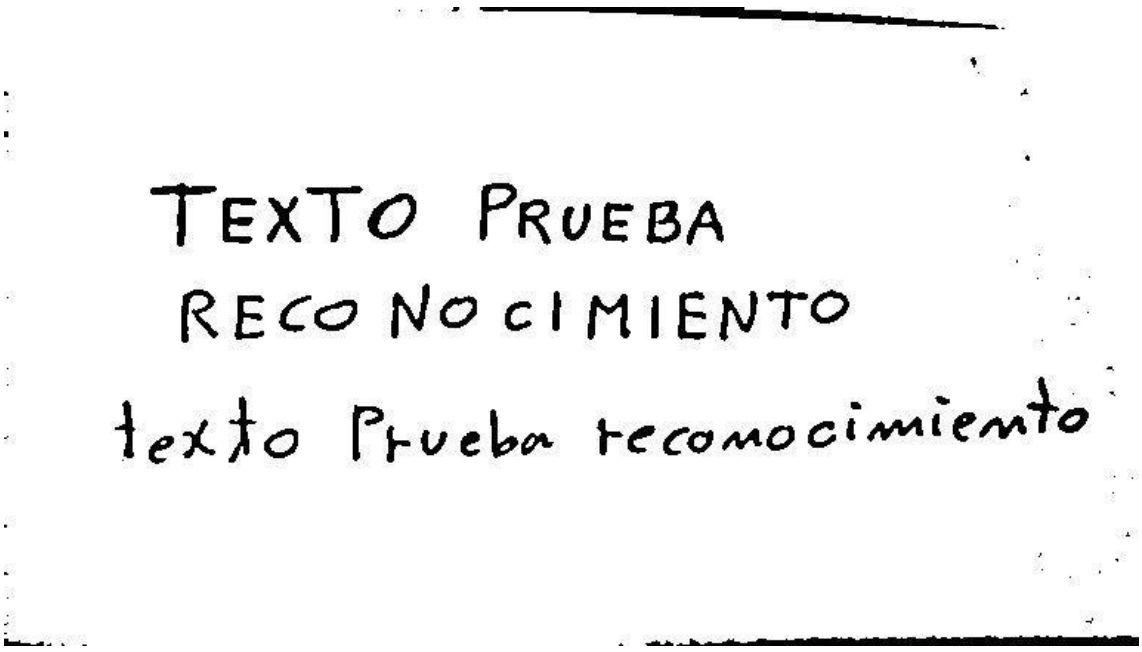

Figura 14 Segunda prueba

En la foto obtenida en la segunda prueba se puede observar el aumento de la calidad de la imagen debido a la eliminación de la cuadricula. Aunque aún se observan errores debidos a la iluminación y a la resolución.

Figura 15 Resultado segunda prueba

En cuanto al resultado, la totalidad del texto en mayúsculas ha sido reconocido, pero el texto en minúscula no ha podido ser reconocido, además, también se puede observar que algunos errores de la imagen los ha reconocido como texto.

La última prueba consistirá en un texto impreso en papel blanco. Se espera que al ser texto monoespaciado el porcentaje de acierto del algoritmo sea mucho mayor.

Figura 16 Tercera prueba

En esta prueba es donde se espera el mayor porcentaje de acierto debido al uso de caracteres monoespaciados y todos los renglones son rectos

í Inserción de las tarjetas y

E encendido del teléfono

Precaucíón: asegúrese de que utiliza una tarjeta SIM
del
tamaño correcto; no recorte la tarjeta SIM y no
utilice un

: adaptador con la misma.

Figura 17 Resultado tercera prueba

Se puede observar que el porcentaje de acierto ha aumentado considerablemente en comparación con las dos anteriores muestras, e incluso se aprecia que no hay muchos caracteres que hayan sido incluidos a causa de errores a la hora de tomar la foto. De todas formas se tendrá que hacer una comprobación humana de los resultados para hacer cualquier tipo de corrección.

7. Futuro de la aplicación

Como se ha ido viendo a lo largo de este trabajo, el proyecto desarrollado está dividido en dos partes principales. La primera parte es la comprensión del funcionamiento del algoritmo de reconocimiento digital de caracteres Tesseract y la segunda parte es el desarrollo de una aplicación para smartphones que explote las funcionalidades de dicho algoritmo para crear digitalizaciones de textos tanto impresos como manuscritos. Al finalizar este trabajo se tiene en poder una primera versión de la aplicación con las funcionalidades básicas de captura, procesado y reconocimiento. No obstante, se espera mejorar la calidad de las características que ya posee y la creación de otras

nuevas para crear un entorno más amigable a la hora de usar esta aplicación. Para versiones siguientes se pretende añadir las siguientes características:

- Corrección de ruido causado a causa de la mala iluminación de la muestra de texto y de aquél incluido por la baja calidad del hardware. Como se ha podido observar en el apartado de resultados, este tipo de ruido confunde al algoritmo Tesseract haciéndole creer que son puntuaciones u otros caracteres que en realidad no existen.
- Para los casos en los que no se disponga de un trípode, o algún mecanismo que pueda mantener estabilizado el móvil, a la hora de tomar la foto cualquier movimiento puede distorsionar el resultado, por ello se pretende incluir una corrección de movimiento para que sea más fácil tomar la foto.
- Añadir la posibilidad de cambiar el método de entrada por archivos que se encuentren en la memoria del dispositivo, para volver a procesar un texto que no se pueda fotografiar o procesar un texto al que no se tenga acceso físico.
- Posibilidad de trabajar con más de un idioma, para abrir las fronteras y que no sea un problema de reconocimiento el idioma en el que esté escrito el texto.

8. Conclusiones

En los tiempos que corren, la inteligencia artificial y la robótica están dejando de formar parte del mundo de la ciencia ficción para ir haciéndose un hueco entre nosotros. El reconocimiento de caracteres cumple un proceso vital en todo este mecanismo de crear robots inteligentes que reconozcan lo que hay en su entorno, lo procesen y puedan actuar respecto a ello.

Tesseract es una herramienta de reconocimiento muy potente que hace un uso muy inteligente de las redes neuronales y el cual todas sus decisiones de diseño llevan revolucionando la tecnología OCR desde el mismo día que se creó. Pero aún con todos estos avances en la materia, las estadísticas hablan y dicen que aún hay margen de mejora. De lo obtenido con las pruebas realizadas de este proyecto se observa que se necesita un alto grado de calidad a la hora de la digitalización de los textos y el porcentaje de acierto en cuanto a los textos manuscritos deja bastante que desear. Todo esto sin contar que los tiempos de procesado por página son bastante altos y hace que todo el proceso sea tedioso. En resumen, se puede decir que a Tesseract aún le quedan unos pocos años para llegar a la cúspide de su potencial, pero esto está lejos de ser una derrota, ya que al igual que cuando se creó la tecnología OCR, solo hace falta un pequeño avance tecnológico para resolver sus problemas.

La aplicación que se ha desarrollado cumple con las expectativas, proporcionando a Tesseract un marco de trabajo donde poder obtener textos digitalizados y mejorados para ser reconocidos. Aunque los resultados están lejos de ser ideales, se puede afirmar que eran esperados teniendo en cuenta que los factores externos a la aplicación eran difíciles de controlar y que se esperaba cierto número de fallos a la hora del reconocimiento, sobre todo en los textos manuscritos. También cabe mencionar los problemas originados por OpenCV y la librería Tess-two, que aun siendo herramientas muy potentes y las mejores para el trabajo que se les ha asignado, están escritas en código C++ y la traducción al entorno Java ha causado problemas más de una vez.

Finalmente, resta por decir que realizar este proyecto ha sido una experiencia más que satisfactoria, viendo como todo esfuerzo invertido en la aplicación hacía que fuera creciendo poco a poco, pasando de no ser nada, a ser una aplicación funcional. Este trabajo ha desarrollado facetas de superación y ha expandido los límites, tanto personales como profesionales, adquiriendo nuevas habilidades y conocimientos en el desarrollo de aplicaciones, como en el campo personal a la hora de la dedicación y de la determinación de terminar este proyecto.

9. Agradecimientos

Este trabajo va dedicado a todos aquellos que han ayudado en su desarrollo, ya sea con conocimientos o con apoyo moral. A mis padres y hermano por su apoyo incondicional durante todos estos años de carrera, a mis amigos por ser uno de los pilares que evitaba que me derrumbara y a mi tutor, que me ha guiado durante todo este trabajo.

10. Bibliografía

[1] Jesús Tomas, Antonio albiol, Mohamed Falhi, Vicente Carbonell *Dispositivos Wearables, Visión Artificial, Google Glass y Android TV.*

[2] Ray Smith, *An Overview of the Tesseract OCR Engine.*

[3] Juan Pablo Ordóñez L. *Reconocimiento Óptico de caracteres (OCR) con redes neuronales estado del arte.*