Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

Aplicación de realidad aumentada para los objetos de un museo

Augmented reality application for museum objects

Alfonso Delgado Vellosillo

Directora: Ana Cristina Murillo

Trabajo Fin de Grado
Ingeniería informática
Computación

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Diciembre 2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Alfonso Delgado Vellosillo,

con nº de DNI 18063916S en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) _____, (Título del Trabajo) Aplicación de realidad aumentada para los objetos de un museo

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 23/11/2018

Fdo: _____

Resumen

En este Trabajo de Fin de Grado, se ha desarrollado una aplicación de realidad aumentada para dispositivos móviles. Esta aplicación se ha desarrollado para ser utilizada en el museo de informática del edificio Ada Byron, ubicado en el campus Río Ebro de la Universidad de Zaragoza. La función de esta aplicación consiste en reconocer los diferentes objetos que alberga el museo, y añadir información sobre su visualización en el móvil en tiempo real.

Para el desarrollo de la aplicación, se plantearon dos tareas principales. La primera era crear un sistema de reconocimiento de objetos lo suficientemente rápido y robusto como para funcionar de forma correcta en un dispositivo móvil, utilizando técnicas de visión por computador. La segunda tarea era la de crear una visualización con información adicional relativa a los objetos detectados.

La mayor parte del trabajo se ha invertido en la creación del sistema de detección y localización de objetos, dejando una visualización de información más sencilla que pueda ejecutarse en móviles de gama media.

Para la creación de la aplicación prototipo, se han implementado tres módulos principales.

El primero es un módulo de reconocimiento de objetos basado en *deep learning*, que mediante el uso de redes neuronales convolucionales (CNNs), es capaz de reconocer diferentes objetos en una imagen. Para el desarrollo de este módulo, se ha utilizado *TensorFlow*, un entorno de desarrollo para la creación y el uso de CNNs. Se compararan diferentes arquitecturas y opciones a la hora de construir una CNN capaz de reconocer los objetos del museo, teniendo en cuenta tanto su corrección en el reconocimiento como su facilidad para integrar dentro de la aplicación móvil, y finalmente se decidió utilizar un modelo existente, *Inception*, re-entrenado para los objetos seleccionados.

La función de este módulo dentro del sistema es la de determinar tres posibles objetos que se encuentren en la imagen para luego localizar uno de ellos de forma más robusta y precisa.

El segundo módulo, es un módulo de reconocimiento basado en el uso de características locales de imágenes (*local features*), cuya función es la de localizar con más precisión en la imagen uno de los objetos reconocidos por el módulo anterior. Este módulo se ha desarrollado utilizando la librería *OpenCV*, en un entorno mixto entre Java (aplicación base) y C++ (módulo). Durante su creación se evaluaron dos tipos de detectores de características (ORB y BRISK), y finalmente, para su correcto funcionamiento en tiempo real se utilizaron los puntos ORB.

El módulo de visualización implementado, utiliza como el módulo anterior la librería *OpenCV*, y aporta una visualización sencilla de información relacionada con el objeto detectado por los módulos anteriores.

Estos tres módulos han sido integrados en la aplicación presentada como prototipo final, y se ha probado su correcto funcionamiento en el museo con un conjunto de quince objetos seleccionados. No se ha podido incorporar todos los objetos porque para cada objeto a añadir es necesario recopilar una gran cantidad de imágenes diferentes (60-70) para re-entrenar la CNN.

Todos los objetivos se han cumplido satisfactoriamente, y se ha conseguido una aplicación funcional y fácil de gestionar. Se ha planteado como trabajo futuro, la mejora del módulo de visualización mediante el uso de herramientas

más complejas para realidad aumentada que se encuentran en desarrollo a día de hoy, como es *ARCore*, pero que actualmente solo funcionan correctamente en móviles de últimas generaciones.

Índice general

Capítulo 1

Introducción

1.1. Motivación y Contexto

La **realidad aumentada** (RA) consiste en añadir información generada de forma virtual, a una imagen real, de tal forma que el resultado final es una imagen que mezcla la realidad con una parte virtual. Esto, permite añadir una capa de información virtual a lo percibimos de manera física del mundo real, y que los usuarios puedan interactuar con esta información, añadiendo una mayor profundidad a la realidad física. Es un campo en auge en los últimos años debido a los grandes avances producidos en campos como la visión por computador o el renderizado de imágenes en tiempo real, y a su vez, a la progresiva mejora del *hardware* disponible para este tipo de aplicaciones. El uso de móviles y otros dispositivos específicos como *headsets* (Holo-lens o Magic Leap One), están facilitado su desarrollo, potenciando la aparición de múltiples aplicaciones de realidad aumentada, tanto en un entorno cotidiano o laboral, como en el mundo del ocio virtual.

Podríamos decir que hay dos grandes grupos de aplicaciones de realidad aumentada, las orientadas a dispositivos específicos como *headsets* o gafas de RA, y las implementadas para dispositivos móviles de propósito más general.

Las primeras requieren de un *hardware* específico, que es utilizado para una visualización más realista y profunda de la que se puede obtener utilizando una pantalla convencional y un ordenador. El segundo grupo de aplicaciones, utilizadas en plataformas móviles, es un un grupo mas heterogéneo, con una gran diversidad de funciones, generalmente orientadas a reconocer objetos/lugares y a proyectar información adicional sobre ellos para aportar alguna utilidad al usuario. Estas últimas se ven restringidas por las limitaciones que supone el uso de dispositivos móviles, ya que la potencia de estos es muy inferior a comparación con los *headsets* y gafas específicamente desarrollados para la realidad aumentada. Aún así, las aplicaciones móviles de realidad aumentada están mucho más extendidas entre la población, ya que no requieren de la compra de un *hardware* específico para su uso. Un ejemplo de estas aplicaciones que se volvió viral de forma reciente es Pokemon GO, la cual logró movilizar a miles de personas, es totalmente gratuita y se puede utilizar en la mayoría de dispositivos *Android* e *iOS*.

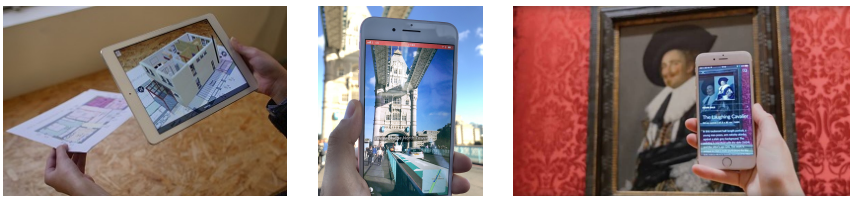
Figura 1.1: Prototipo del proyecto. En el se puede observar el planteamiento básico de la aplicación: apuntar con el móvil a los objetos del museo; el objeto es reconocido y localizado; se añade una imagen u otra información aumentada relacionada alineado con el objeto en la visión del mismo en el móvil.

En este proyecto, nos centramos en este segundo tipo de aplicaciones, para desarrollar una aplicación robusta y dinámica, capaz de reconocer objetos de un museo en tiempo real, con escalabilidad para objetos nuevos, o incluso a otros ámbitos o museos. Además, el fácil funcionamiento de la aplicación debe hacer que sea accesible a todos los usuarios y que pueda ser introducida de cara a todo tipo de públicos. La Figura 1.1 muestra un esquema de lo que se propone en esta aplicación. En ella, se puede observar que el planteamiento de la aplicación consiste en enfocar con el móvil a un objeto, encuadrándolo en el centro de la pantalla, y a continuación reconociendo automáticamente de que objeto se trata y localizando su posición en la imagen, se coloca información adicional del objeto sobre la posición en la que sea detectado.

La motivación para desarrollar este proyecto fue crear una aplicación específica para identificar los objetos almacenados en el museo de informática de la Universidad de Zaragoza[1]. Una motivación extra en este trabajo es la profundización en temas relacionados con la computación, como pueden ser el *deep learning* y *computer vision*, que son dos campos más generales y de gran interés hoy en día, con muchas y diversas aplicaciones en el campo de la computación, y no solo orientados a la realidad aumentada. Por ello, se ha valorado muy positivamente en este trabajo, el poder trabajar con estos dos campos, y así poder profundizar en ellos de cara a su aplicación en otros futuros proyectos y adquirir experiencia en estos campos.

Productos existentes/similares relacionados. Actualmente, existen múltiples aplicaciones relacionadas y con similares objetivos a este proyecto. La mayoría basan su funcionamiento en el reconocimiento en la escena de objetos de una base de datos de elementos de referencia conocidos, sobre los cuales muestra información superpuesta una vez son localizados en la escena que esta enfocando el usuario con su móvil, para crear experiencias de realidad aumentada. Por

[1] `http://mih.unizar.es`



(a) (b) (c)

Figura 1.2: Ejemplos recientes de aplicaciones de RA en móviles. (a) Aplicación *Augment* (fuente: `https://www.augment.com/`). (b) Aplicación *AR City* (fuente: `https://www.blippar.com`). (c) Aplicación Smartify en funcionamiento en un museo. (fuente `https://smartify.org/`)

ejemplo, de cara a la realidad aumentada en museos, encontramos la aplicación *Smartfy*, una aplicación desarrollada para ser utilizada en museos, capaz de reconocer un objeto y a continuación mostrar información de dicho objeto, como una breve descripción y una imagen, tal y como se ve en la Figura 1.2 (c).

En la Figura 1.2 (a) y (b) vemos otros dos ejemplos de aplicaciones de realidad aumentada recientes. La aplicación *Augment* es una aplicación basada en la detección de imágenes, para luego poder añadir visualizaciones complejas en 3D de datos sobre el contenido de las imágenes reconocidas. Otra aplicación que puede mostrar hasta donde puede llegar la realidad aumentada es *AR City* (Figura 1.2 (b)). Esta aplicación añade información en tiempo real sobre localizaciones, rutas y demás elementos que pueden aparecer en el día a día en una ciudad.

Además, existen muchas herramientas para desarrollo de aplicaciones de realidad aumentada como*ARCore* [2] o ArKit [3] entre otros, que proporcionan entornos muy completos con las funcionalidades básicas de detección de la información 3D del entorno incluidas.

Contexto de desarrollo del proyecto. Este proyecto se ha realizado en colaboración con el grupo de investigación de robótica de la Universidad de Zaragoza. El objetivo es crear una herramienta informativa para el museo de informática del edificio Ada Byron, ubicado en el campus río Ebro, en Zaragoza. El proyecto realizado, tiene aplicación directa, de cara a que cualquier visitante o persona interesada en alguno de los objetos del museo, fuera capaz de identificarlos y obtener información de ellos, de forma rápida e interactiva, sin la necesidad de añadir una descripción física, lo cuál es una gran ventaja en algunos objetos en concreto, debido a la dificultad de añadir una descripción física al objeto al no haber suficiente espacio en las vitrinas para colocar una redacción de todos los objetos. Además, al ser información que se ha añadido de forma virtual, esta pueden sen elementos multimedia, como vídeos, o fotografías, en vez de contener únicamente texto plano.

Como entorno de desarrollo se ha utilizado *Android* para el desarrollo base de la aplicación, debido a su gran accesibilidad y a su extensión entre los usuarios. Además, para el desarrollo de los diferentes módulos que conforman el sistema, se han utilizado dos librerías: *OpenCV* para reconocimiento basado en

[2] `https://developers.google.com/ar/`
[3] `https://developer.apple.com/arkit/`

local features, y *TensorFlow* para el reconocimiento de objetos utilizando *deep learning*.

1.2. Objetivos y Tareas

Como se ha comentado anteriormente, el objetivo general de este proyecto es realizar una aplicación móvil de realidad aumentada que reconozca diferentes objetos del museo de informática alojado en el edificio Ada Byron. Una vez reconocido un objeto, la aplicación muestra información relativa al objeto reconocido.

Los principales problemas a abordar en este proyecto son:

- La obtención de un sistema basado en la visión por computador lo suficientemente rápido y robusto como para tener una aplicación para dispositivos móviles que reconozca los objetos de forma robusta y en tiempo real.
- La representación de la información asociada a cada objeto reconocido de forma gráfica en tiempo real, en función de la posición del objeto real.

Para llegar a estos objetivos, se plantearon las siguientes tareas a realizar durante el proyecto:

- Instalación del entorno de desarrollo y librerías necesarias para el proyecto.
- Desarrollo del módulo de reconocimiento de objetos.
- Desarrollo del módulo de visualización, utilizando la información proporcionada por el módulo de reconocimiento.
- Pruebas e integración de todos los módulos implementados.
- Documentación del proyecto y desarrollo de la memoria.

Cronograma del desarrollo de las tareas. La distribución de las tareas y objetivos a lo largo del tiempo está representado en el diagrama de la Figura 1.3.

1.3. Resumen de la memoria

La estructura del resto de capítulos de esta memoria, es la siguiente:

1. Capítulo 2. Trabajo relacionado: Información teórica sobre las técnicas utilizadas en este proyecto y su aplicación práctica.
2. Capítulo 3. Sistema propuesto: Breve descripción del funcionamiento del sistema y de sus partes.
3. Capítulo 4. Reconocimiento basado en *deep learning*: Explicación en profundidad del módulo que implementa una red neuronal para reconocimiento de objetos, y de las pruebas realizadas a dicho módulo.

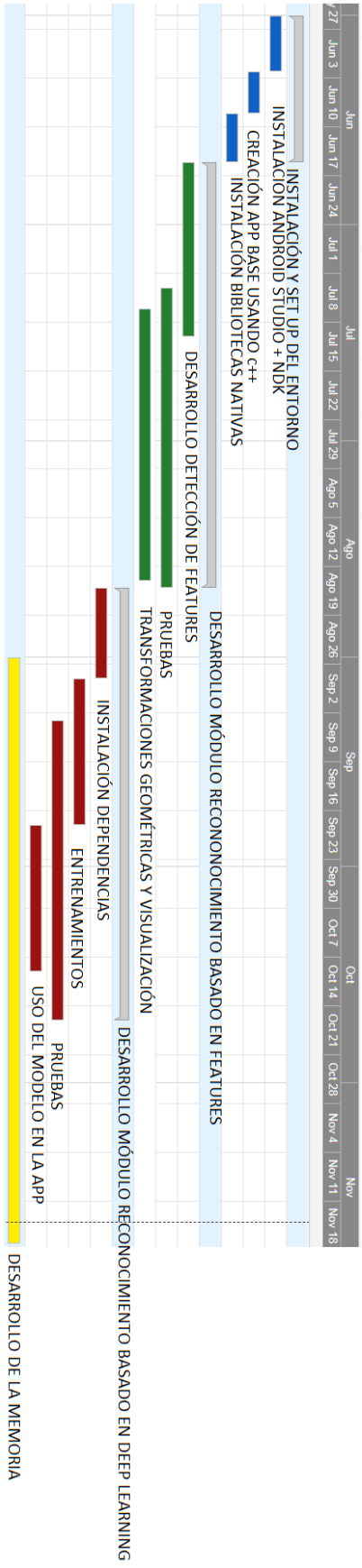


Figura 1.3: Diagrama de Gantt donde se representan las tareas realizadas a lo largo del tiempo.

4. Capítulo 5. Reconocimiento basado en *local features*: Descripción en profundidad del módulo que implementa el reconocimiento de objetos basado en *computer vision*, y de las pruebas realizadas a dicho módulo.

5. Capítulo 6. Prototipo construido: Explicación en profundidad sobre el sistema completo implementado, entrando en detalles técnicos de su implementación concreta y del uso de los módulos anteriormente explicados.

6. Capítulo 7. Conclusiones: Conclusiones finales sobre el sistema propuesto y los resultados obtenidos.

7. Anexos A, B, C con los resultados detallados de las pruebas realizadas al módulo de reconocimiento basado en *local features*, la descripción en detalle de la base de datos de imágenes utilizada y un manual para añadir nuevos objetos a la aplicación, respectivamente.

Capítulo 2

Trabajo Relacionado

Este capítulo resume las técnicas principales utilizadas para el sistema de detección y reconocimiento visual automático realizado en este proyecto. En la literatura relacionada con reconocimiento visual automático encontramos dos grandes grupos de soluciones: las técnicas basadas en características locales de imagen, y técnicas basadas en un análisis conjunto de toda la imagen. De este segundo grupo cabe destacar por su gran avance en los últimos años, aquellas basadas en modelos de *deep learning*.

2.1. Reconocimiento con características locales

Las técnicas de reconocimiento basadas en la detección y búsqueda de correspondencias de características locales, o *local features*, en las imágenes han demostrado muy buenos resultados en reconocimiento visual en las últimas décadas. [1].

La detección de estas *features* en imágenes consiste en la búsqueda de puntos que contengan ciertas características representativas en la imagen. Posteriormente, es común comparar estos puntos característicos extraídos de una imagen con los extraídos de otra para, si las correspondencias son suficientes y consistentes, determinar si ambas imágenes contienen el mismo objeto o escena.

2.1.1. Características locales

En la literatura de visión por computador encontramos muchísimas propuestas para extracción de características locales, con especial impacto en los últimos años encontramos las propuestas de SIFT [1], o SURF [2], que fueron de las primeras en resolver de manera efectiva problemas de invarianza a escala y perspectiva, permitiendo sistemas más robustos y realistas. En particular, en este trabajo nos interesan las características que están orientadas a procesados rápidos, es decir, reducir el tiempo de procesado en cada imagen, aunque a veces sea a costa de bajar algo la precisión. En el trabajo *Comparative Evaluation of Binary Features* [3], encontramos una comparativa de distintas características binarias, donde se resumen ventajas e inconvenientes de los algoritmos mas utilizados. En este trabajo vamos a evaluar el uso de los dos detectores de puntos y descriptores, más conocidos y utilizados en la literatura, *ORB* [4] y *BRISK* [5].

Dado que ambos se basan en utilizar una representación binaria de los descriptores, ofrecen un menor coste computacional, manteniendo un funcionamiento robusto.

Tanto *BRISK* como *ORB*, se basan en en descriptores binarios pero tienen varias diferencias. Además de estrategias un poco distintas para encontrar los puntos de interés de una imagen, sobre todo se diferencian a la hora de calcular el descriptor de cada uno de estos puntos, cambiando la forma en la que se interpretan y detectan las esquinas de la imagen. El descriptor en ambos casos será un vector de valores binarios. Cada valor binario se calcula verificando si ciertos pares de puntos de cierto patrón son iguales o distintos. La Figura 2.1 muestra el patrón utilizado para seleccionar pares de puntos en cada algoritmo.

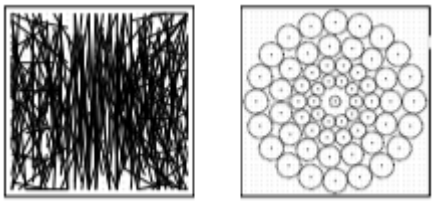


Figura 2.1: Patrón con los pares de puntos utilizados para calcular el descriptor de ORB (derecha) y BRISK (izquierda) (fuente de imagen: [3])

2.1.2. Correspondencias Robustas utilizando Homografía y Matriz Fundamental

Una vez realizada la extracción de las características, se suele realizar una búsqueda de correspondencias en dos fases.

En una primera se establecen las **correspondencias (o emparejamientos) *iniciales***, utilizando técnicas de búsqueda del vecino mas cercano, o *nearest neighbour*. La opción básica consiste en simplemente buscar, para cada descriptor de una imagen, cual de entre todos los descriptores que hay en la segunda imagen es más parecido. Una versión mejorada verificando los dos vecinos más cercanos es más frecuente y ha dado mejores resultados [1]. Esta técnica de búsqueda del vecino más cercano sigue la siguiente fórmula para decidir si una correspondencia es buena:

$$Correccion(i) = Emparejamiento1(i) < Emparejamiento2(i) * 0{,}7. \quad (2.1)$$

En la ecuación anterior, i representa uno de los posibles emparejamientos entre un punto de una imagen, y otro punto de la otra imagen. $Correccion$ de i representa si un emparejamiento se considera bueno o no, siendo sus posibles valores 0 (erróneo) o 1 (pasa la criba). $Emparejamiento1$ representa la distancia al descriptor del mejor emparejamiento del punto i con otro punto de la otra imagen, y $Emparejamiento2$ representa la distancia al segundo mejor emparejamiento del punto i con otro punto de la otra imagen. Si el mejor emparejamiento es lo suficientemente mejor que el segundo (en este caso se ha puesto un límite, que es el valor del $Emparejamiento2$ multiplicado por 0,7), se considerará que el

*Emparejamiento*1 es correcto.

Si es necesario un **conjunto** más fiable y **robusto** de correspondencias, es común realizar una segunda fase, donde se realiza una verificación geométrica utilizando restricciones conocidas entre dos imágenes. Estas restricciones entre dos imágenes de una misma escena se conocen como *Homografía* y *Matriz Fundamental*. Más detalles de estas dos construcciones en [6], a continuación incluimos un breve resumen.

Homografía. La Homografía [6] es una matriz $3x3$ que relaciona la proyección en dos imágenes distintas de puntos que son co-planares en la escena real. Esta matriz **H** codifica la posición relativa entre las dos fotos adquiridas de la misma escena y relaciona las proyecciones de un punto en 3D en cada una de las dos imágenes (**x** y **x'**) directamente:

$$\mathbf{x'H} = \mathbf{x} \tag{2.2}$$

En el sistema construido, la homografía será utilizada para conocer las transformación entre dos fotografías de un mismo objeto (plano), pudiendo así ubicar puntos de una fotografía en la otra. Como se ha comentado, la homografía es eficaz para relacionar geometrías planas, pero no sucede objetos o escenas generales en 3D.

Matriz Fundamental. La matriz fundamental **F** [6], es una matriz de $3x3$ que relaciona los puntos correspondientes en dos imágenes, al igual que la homografía, pero en un caso más general. Como se puede observar en la Figura 2.2, Utilizando coordenadas homogéneas, dada la representación de dos puntos **x** y **x'**, que son puntos correspondientes en la primera y segunda imagen respectivamente, **Fx** describe una recta, llamada *línea epipolar* en la segunda imagen donde se encuentra el punto **x'**, De esta forma, la expresión que relaciona todos los pares de puntos de ambas imágenes mediante la matriz fundamenta F es:

$$\mathbf{x'}^T\mathbf{Fx} = 0 \tag{2.3}$$

Correspondencias robustas. Tanto la homografía como la matriz fundamental, se pueden utilizar para verificar cuales de las correspondencias entre los puntos emparejados son correctas. En ambos casos, dado un conjunto de correspondencias *iniciales*, y realizando una estimación robusta de la H o la F, podemos comprobar que pares de puntos emparejados se pueden considerar correctos y cuales no, ya que los correctos serán consistentes con la estimación de estas restricciones geométricas.

Realizando una estimación robusta (por ejemplo con RANSAC [6]) de la H o la F, podemos ver que pares de puntos encajan con dicha restricción (*inliers*), o no. Todos los *inliers* de una estimación corresponden con puntos que han sido emparejados de forma correcta y que resultan consistentes con la geometría de la escena, mientras que el resto de puntos emparejados se descartan.

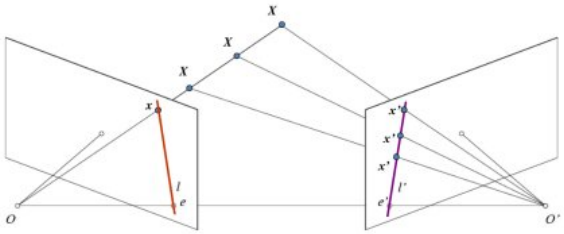


Figura 2.2: Geometría Epipolar. Se puede apreciar que un punto en una imagen, es una recta en otra imagen tomada desde un origen diferente. De esta forma, la matriz fundamental relaciona los puntos de una imagen con los de otra mediante las líneas epipolares. **o** y **o'** se corresponden con los puntos desde los que se han tomado las imágenes, **x** es el punto observado desde el origen **o**, **x'** son los puntos pertenecientes a la línea creada por el punto **x** en el origen **o'**, y **e** y **e'** corresponden al mismo punto observado desde **o** y **o'**. (Fuente: Wikipedia - Geometría epipolar)

2.2. Reconocimiento basado en *deep learning*

El reconocimiento de imágenes basado en técnicas de *deep learning* estudiado se basa en técnicas de aprendizaje supervisado, es decir, entrenar un modelo a partir de imágenes de ejemplo etiquetadas.

La principal diferencia en el planteamiento de este tipo de reconocimiento comparado con el reconocimiento basado en *features* es que en este segundo el reconocimiento se basa en unos cuantos puntos característicos a emparejar, y utilizando *deep learning* el reconocimiento se hace a nivel de todos los píxeles de una imagen, y no utilizando sólo unos puntos concretos. Además, el mayor coste computacional de la red neuronal se encuentra en su entrenamiento, pero su uso posterior para predicciones es muy rápido, por lo que el uso de estas redes presenta una mayor escalabilidad que el uso de *local features*.

Usando *local features*, normalmente, la comparación entre imágenes resulta lineal con el número de imágenes del modelo (comparamos todas las *features* de la imagen actual con todas las *features* extraídas de las imágenes de referencia). Y por lo tanto, también crece conforme añadimos mas clases o imágenes de ejemplo. Sin embargo, en el caso de usar una red neuronal, necesitamos más datos de ejemplo, y por lo tanto más tiempo a la hora de entrenar el modelo, pero no tiene prácticamente coste adicional a la hora de su ejecución.

En el sistema propuesto en este trabajo, se ha utilizado un tipo concreto de redes neuronales profundas, conocidas como redes convolucionales (*convolutional neural networks*), que son las más adecuadas para trabajar en clasificación y reconocimiento de imágenes hoy en día. A continuación se resumen los conceptos básicos relacionados con diferentes técnicas de reconocimiento basadas en *deep learning*.

Redes neuronales profundas - *Deep learning*. Las redes neuronales, son un modelo computacional basado en el uso de pequeñas unidades denominadas neuronas artificiales. Estas neuronas artificiales, reciben unos datos de entrada, y generan otros datos datos de salida, en función de los recibidos. Así, para

formar una red neuronal, nos encontramos con que como se ve el la Figura 2.3 (izquierda), las salidas de unas neuronas se conectan a las entradas de otras, creando así capas de neuronas interconectadas ocultas (*Hidden*), y una capa final (*Output*), donde se encuentran las salidas de la red.

Cada enlace entre 2 neuronas, se encuentra ponderado por un peso específico, cuya función es la de aumentar o inhibir la salida de dicha neurona, haciendo así posible regular que neuronas son más influyentes o menos en el resultado final de la red. A su vez, cada neurona contiene la denominada **función de activación**, cuya utilidad es servir como umbral, que modifica el valor resultado o impone un límite que se debe sobrepasar antes de propagarse a otra neurona, de tal forma que si la función de activación de una neurona no se activa, las salidas de dicha neurona tampoco se activarán.

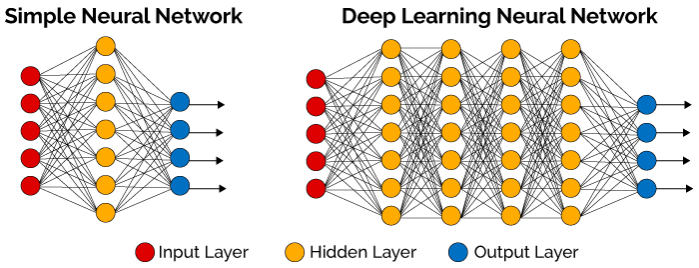


Figura 2.3: Una red neuronal simple (izquierda), frente una red neuronal profunda(derecha). (fuente: `https://planetachatbot.com/deep-learning-facil-con-deepcognition-9af43b2319ba`)

En los últimos años, están teniendo mucho auge y buenos resultados las redes neuronales profundas, o *deep neural networks* [7], que de manera simplificada podríamos decir que son una red convencional pero con muchas capas ocultas, como en el esquema de la Figura 2.3 (derecha).

Uno de los problemas principales que plantea el uso de redes neuronales, es la necesidad de una gran cantidad de datos, para realizar su entrenamiento. El entrenamiento de una red neuronal es el proceso por el cual se ajustan los pesos de las conexiones entre las neuronas, para obtener la salida deseada al introducir nuevos datos al sistema. En este proceso se introducen datos etiquetados en la red neuronal, que va ajustando sus pesos en función de si las salidas obtenidas para las imágenes de entrenamiento son correctas o no.

En particular, en este proyecto se utilizan redes neuronales convolucionales (CNN), detalladas a continuación.

Redes neuronales convolucionales - CNN Las redes neuronales convolucionales, son un tipo de red neuronal. La operación básica de este tipo de redes es la convolución sobre matrices bidimensionales, operación ampliamente utilizada en el manejo tradicional de imágenes. Esto es lo que las hace especialmente útiles a la hora de clasificar imágenes, ya que se ha comprobado que realizan operaciones similares a las que realizan muchos de los algoritmos tradicionales de procesamiento de imágenes.

Como se puede ver en la Figura 2.4, normalmente, las primeras capas de una CNN son capas convolucionales (*Convolutional Layers*), que aplican sus operaciones sobre matrices de dos dimensiones, que representan secciones de la

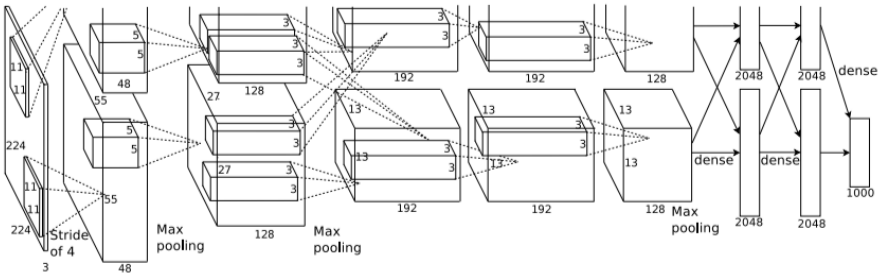


Figura 2.4: Composición de las capas de la CNN AlexNet [8]. Se pueden ver las primeras capas que son *Convolutional layers* que trabajan con los datos de la imagen, y las capas de *pooling* que reducen la magnitud de estos datos. Finalmente, las últimas capas, son *Dense layers*, que generan la salida final del modelo.

imagen, extrayendo de manera automática las características de las imágenes (según los patrones que se han aprendido durante el entrenamiento). Las capas finales de la red, conectan todas las salidas de las capas convolucionales y por ello se conocen como *Dense Layers*, y se encargan de la clasificación final de las características obtenidas anteriormente. Además de estas dos capas, estas redes tienen otro tipo de capas intermedias, conocidas como *Pooling Layers* cuya función es la de agrupar los dato de salida de la capa anterior, manteniendo su información pero reduciendo su volumen, para que los datos que lleguen a las siguientes capas y sean procesados en un menor tiempo.

En las CNN, los datos de entrenamiento necesarios son imágenes etiquetadas de los objetos que se quieren reconocer. Esto plantea el problema de que para aumentar el número de clases se requieren, se necesitan imágenes de los nuevos objetos a reconocer, y cuanto mayor sea el número de imágenes más preciso será el sistema. Otro dato relevante sobre las imágenes a utilizar en los entrenamientos, es que las imágenes deben tener cierto grado de variación, ya que si las imágenes son muy similares entre sí, o el objeto se encuentra siempre con el mismo fondo, el sistema al aprender características de toda la imagen, aprenderá características no pertenecientes al objeto, reconociendo el fondo de la imagen como objeto por ejemplo, o si las imágenes son muy similares entre sí, el modelo no sería capaz de reconocer un objeto en una imagen en la que haya pequeñas variaciones con respecto a las imágenes usadas en el entrenamiento. A esto se lo conoce como sobre ajuste.

Capítulo 3

Diseño del Sistema Propuesto

3.1. Módulos del sistema

Esta sección presenta una vista global de todos los módulos desarrollados, que conforman la aplicación, y de como se interconectan entre sí para formar el prototipo de aplicación móvil capaz de realizar un reconocimiento automático de objetos y visualizar información relacionada con los objetos reconocidos en tiempo real.

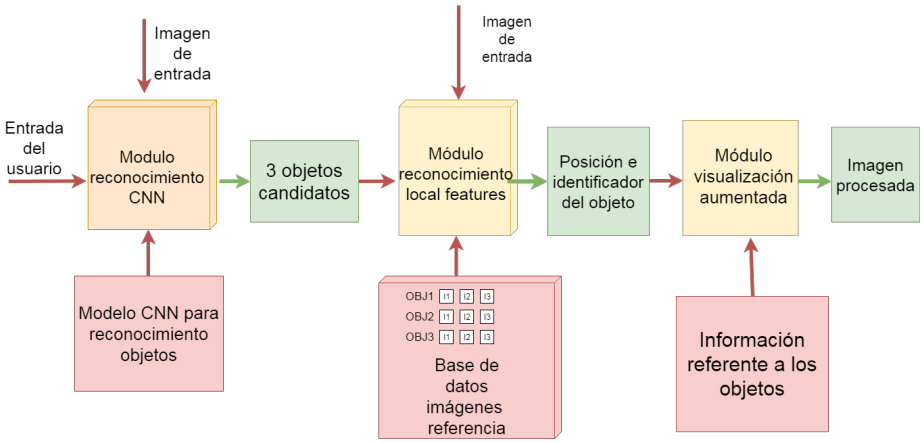


Figura 3.1: Componentes del sistema de visualización de realidad aumentada de información de objetos. Cada módulo (en amarillo) tiene una parte del modelo almacenado correspondiente (en rojo) con los datos de referencia necesarios para ejecutarlo. Las flechas de color verde indican la sida de cada módulo, y a su vez las flechas rojas indican las entradas a cada módulo.

Los **componentes principales** del sistema diseñado se conectan como muestra la Figura 3.1. Los módulos desarrollados para el reconocimiento automático y la visualización en la aplicación de realidad aumentada son los si-

guientes:

- Módulo de reconocimiento CNN (basado en *deep learning*).
- Módulo de reconocimiento *local features* (basado en *local features*).
- Módulo de visualización aumentada.

Además de estos tres módulos, el sistema está compuesto por otras tres partes necesarias para su funcionamiento que se detallan a continuación:

- Modelo CNN: Modelo reentrenado para los objetos del museo a reconocer, utilizado en la primera fase del reconocimiento (reconocimiento CNN), para procesar una imagen informando de a que tres objetos de referencia es más similar.
- Base de datos imágenes de referencia: Base de datos compuesta de tres imágenes de cada objeto de referencia. Se almacenan tres imágenes de cada objeto para tener una visión frontal y de los dos laterales desde los que el objeto puede ser observado.
- Información referente a objetos: Base de datos que almacena tres imágenes con información adicional de algunos objetos de referencia, para ser mostradas en el módulo de visualización aumentada, y el nombre de todos los objetos.

El funcionamiento general del sistema al ejecutarse en el móvil se puede resumir en el siguiente **flujo de ejecución**:

1. Inicializar la aplicación, cargando la información de referencia, siendo esta el modelo reentrenado usado por la CNN, los descriptores ORB de las imágenes de referencia, y la información de interés relacionada con los diferentes objetos a clasificar.
2. Tomar una imagen a través de la cámara del dispositivo.
3. Clasificar la imagen tomada usando *deep learning*.
4. Localizar el objeto, usando visión por computador, de entre los candidatos seleccionados en el paso anterior por el módulo de *deep learning*.
5. Visualizar la información en pantalla utilizando la localización del objeto anteriormente calculada.

Los siguientes capítulos describen en detalle cada uno de los diferentes módulos que conforman el sistema.

3.2. Entorno de desarrollo del sistema

Esta sección resume el entorno de desarrollo y las herramientas utilizadas para desarrollar el proyecto, y explica porque se han seleccionado dichas herramientas.

Para este proyecto se decidió realizar una aplicación móvil ***Android***, para lo cual se utilizó su entorno de desarrollo estándar, **utilizando tanto Java**

como C++. Se ha trabajado con el *Android Strudio NDK*, debido a la facilidad que da para enlazar código creado con Java y C++. De esta forma, se pueden crear funciones que utilicen elementos hardware del dispositivo *Android*, como la cámara o el almacenamiento, de forma fácil utilizando Java, y a su vez enlazarlo con funciones creadas en C++, donde resulta mas sencillo y eficaz programar, y es capaz de dar soporte a tareas más complejas y costosas como procesamiento de emparejamientos entre descriptores de imágenes, o el cálculo de las matrices fundamentales y de homografía.

Este aspecto era fundamental, debido a que se querían realizar tareas de reconocimiento y localización de objetos, que implican algoritmos complejos de visión por computador, para los cuales es necesario utilizar la librería *OpenCV*, cuya versión de C++ es mucho más completa y eficiente que su versión de Java. Una vez configurado el NDK de forma correcta, se comprobó su correcto funcionamiento creando una aplicación básica de Java, la cual ejecutara una función escrita en C++, y se procedió a añadir la librería *OpenCV* dentro de la aplicación.

Una vez instalados y configurados todos los componentes básicos (obtención de imágenes de la cámara, módulo de prueba de librería *OpenCV* sobre C++), se realizó un primer prototipo "vacío"de la aplicación para comprobar el correcto funcionamiento y conexión entre la estructura base de la aplicación implementada en Java, y el módulo de reconocimiento de imágenes basado en *local features*, implementado en C++. En esta aplicación, la captura de imágenes de la cámara se realiza mediante funciones de *Android* en Java, y utilizando la versión de Java de *OpenCV* para transformar la imagen a un formato reconocible por *OpenCV* en C++, en concreto se transforma la imagen del formato *bitmap* (*Android*), a una matriz de *OpenCV*. La imagen ya transformada es editada mediante funciones implementadas en C++ utilizando *OpenCV*, y finalmente se vuelve a transformar de forma inversa la imagen para mostrarla por pantalla utilizando las librerías del sistema *Android*, implementadas en Java. Todo este comportamiento se puede apreciar en la Figura 3.2.

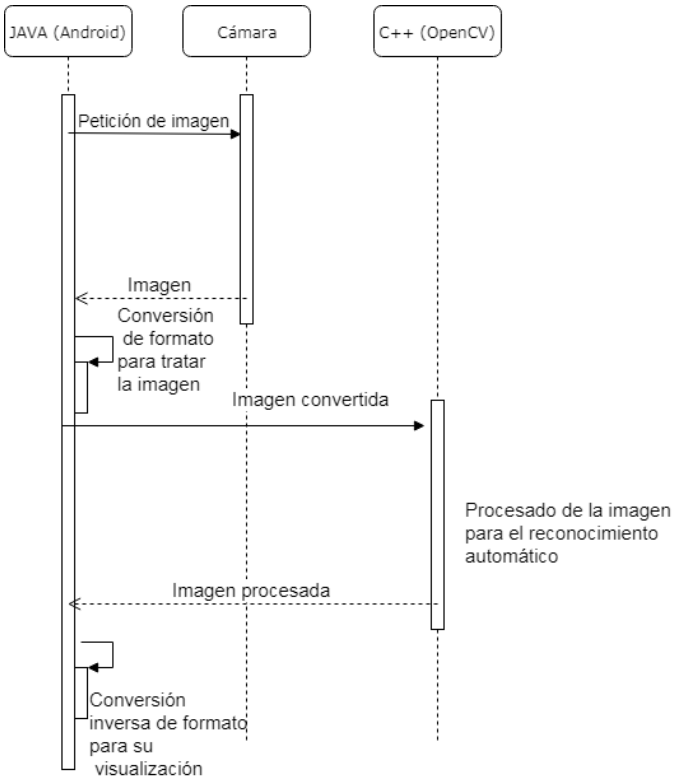


Figura 3.2: **Proceso de adquisición y procesado de una imagen** probado en la estructura de base de la aplicación. La captura de una foto se arranca desde Android, que realiza una petición a la cámara. La imagen se convierte para pasar al módulo de reconocimiento basado en *local features* que está implementado en C++, y cuando este termina, devuelve la imagen final a mostrar al módulo de Android, donde se vuelve a convertir, para que se ejecute la visualización.

Capítulo 4

Módulo de reconocimiento de objetos basado en CNNs

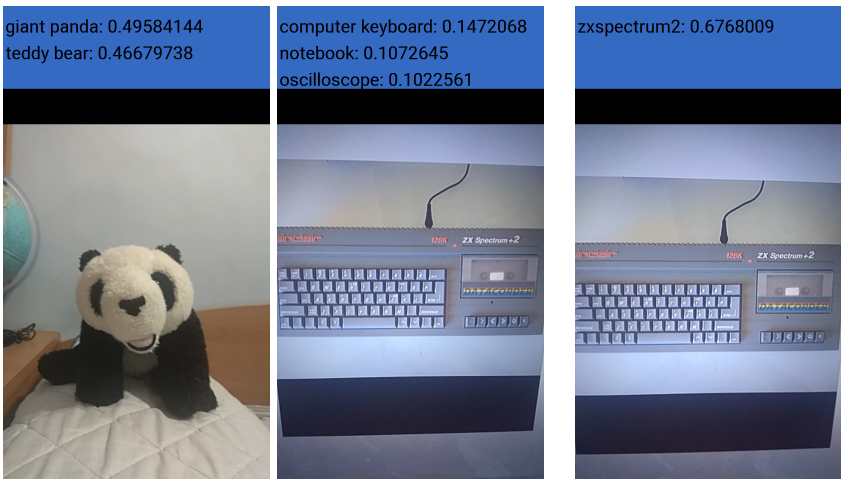
Este capítulo detalla el módulo desarrollado para el reconocimiento de objetos mediante técnicas de *deep learning*, y evalúa la corrección de sus elementos principales. La función de este módulo, es la de reconocer una imagen, y proporcionar los tres objetos pertenecientes al conjunto de objetos de referencia, con una mayor probabilidad de aparecer en la imagen.

4.1. Construcción de la CNN para reconocer los objetos del museo

Por las ventajas de escalabilidad comentadas anteriormente, junto con los buenos resultados demostrados en la literatura reciente, las técnicas basadas en redes neuronales resultan las más eficaces para realizar un reconocimiento inicial de que objetos hay en la imagen. En particular, este módulo está diseñado para detectar que tres clases son las más similares a la imagen obtenida por la cámara, para posteriormente utilizar esas tres clases como posibles candidatos a comparar en el módulo de reconocimiento basado en *local features*.

Entorno de desarrollo de *deep learning* en *Android*. Para realizar el reconocimiento de los diferentes objetos se ha utilizado el entorno de desarrollo *TensorFlow* [9], el cuál provee de las herramientas necesarias para diseñar y utilizar redes neuronales convolucionales. En este caso, se ha utilizado la versión de *TensorFlow* específica para *Android* [10]. Se ha utilizado para reentrenar una red neuronal convolucional capaz de reconocer diferentes objetos, y que a su vez pudiera funcionar sobre dispositivos *Android*. Para poder hacer funcionar *TensorFlow* en un dispositivo Android se ha seguido el ejemplo de la aplicación *Android* de clasificador de objetos de *TensorFlow*.

Selección de la arquitectura de CNN a utilizar. En el entorno de *TensorFlow* encontramos numerosas arquitecturas de CNNs para clasificación de imágenes. De entre ellos, se han considerado las arquitecturas de *Inception* [11] y de *MobileNet* [12], el primero de ellos se eligió debido a sus buenos resultados



(a) (b)

Figura 4.1: Resultados de clasificación de imágenes con un modelo estándar entrenado en *ImageNet* (a), y resultados más concretos que se consiguen con nuestro modelo específico para el museo (b). En ambos casos las capturas de pantalla corresponden con una aplicación de prueba que carga dichos modelos en Android.

y su uso extendido, y el segundo se eligió debido a que está orientado hacia aplicaciones móviles, como la desarrollada.

Tanto *Inception*, como *MobileNet* han sido entrenadas para reconocer las 1000 clases de objetos de un *dataset* público muy conocido para reconocimiento de objetos, *Imagenet* [13].

Imagenet es una base de datos de imágenes que es utilizada muy frecuentemente en materia de visión por computador, como banco de pruebas. En esta base de datos se encuentran almacenadas y etiquetadas una gran cantidad de imágenes de objetos diferentes.

En ambos casos, se ha modificado levemente el modelo para poder entrenarlo en el conjunto de clases de interés para nuestro trabajo, en lugar de las clases por defecto que tenían dichos modelos. Como se puede ver en la Figura 4.1 (a), los modelos disponibles suelen ser genéricos, y reconocen objetos tipo como pueden ser una televisión o un oso de peluche. Sin embargo, nuestro objetivo, como se ve en la Figura 4.1 (b), es distinguir clases mucho más concretas, dejando a un lado clases como ordenador, y utilizando como clases modelos concretos de ordenadores (ZXSpectrum 2, Apple 2,...).

Realizando diferentes entrenamientos con estas dos arquitecturas, detallados más adelante, se llegó a la conclusión de que la mejor opción de arquitectura era *Inception*, debido a que la precisión del modelo usando dicha arquitectura era mejor que utilizando *MobileNet*. Esto es fácilmente apreciable en las Figura 4.2 y Figura 4.3, donde se pueden apreciar los valores de la precisión en los entrenamientos con ambas arquitecturas.

Entrenamiento de la CNN y base de datos de entrenamiento. Una vez elegida la arquitectura de CNN a utilizar, como se ha comentado, no se

Objeto	Nº Imágenes Museo	Nº Imágenes Internet
AmstradCPC	5	55
Apple2	14	46
Atari520ST	8	52
Atari800XL	5	55
Commodore64	7	53
Commodore Amiga	12	49
IBM PowerPC	20	42
Macintosh ED	13	47
ShackTRS	3	58
Sinclair QL	4	56
ZXSpectrum ULA	22	38
Toshiba 64k	6	54
ZXSpectrum	6	55
ZXSpectrum 2	11	49
ZXSpectrum 81	6	54

Tabla 4.1: Distribución de las diferentes imágenes recogidas para realizar los reentrenamientos de la CNN.

ha entrenado un modelo desde cero, sino que se han reentrenado las últimas capas de dichos modelos ya entrenados para la detección de objetos. Esta técnica, conocida como *fine tunning*, permite reaprovechar el entrenamiento de un modelo entrenado con una gran cantidad de imágenes de ejemplo etiquetadas, utilizando únicamente imágenes de los nuevos objetos a reconocer. Esto evita la necesidad de tener una gran cantidad de datos para poder entrenar un modelo de este tipo desde cero.

Para realizar este *fine-tuning* se han utilizado funcionalidades estándar de *TensorFlow*. El *script* utilizado requiere imágenes etiquetadas de cada objeto que queremos reconocer. Para ello se ha recolectado una nueva **base de datos**, que consiste en conjuntos de entre 60 y 70 imágenes diferentes por cada objeto a reconocer. La Tabla 4.1 muestra un resumen de la base de datos de imágenes recopilada. Entre estas imágenes se encuentran fotografías realizadas con diferentes teléfonos móviles a los objetos en el museo, pero a su vez, la gran mayoría de las imágenes utilizadas en el entrenamiento, se han obtenido de internet como se puede ver en la Tabla 4.1, para de esta forma evitar el sobre ajuste que se podría producir al entrenar con fotografías demasiado similares capturadas de cada objeto en una misma escena/entorno, de tal forma que el modelo tomara el fondo también como parte del objeto.

Se eligieron estos objetos para poder comprobar el funcionamiento de la aplicación con una vitrina central completa, ya que once de los objetos se encuentran en la misma vitrina, y se añadieron otros cuatro, dos pertenecientes a vitrinas laterales y dos pertenecientes a diferentes vitrinas centrales, para así poder probar su funcionamiento con una mayor variedad de vitrinas.

Además, cabe destacar que durante el reentrenamiento (*fine-tuning*) de la red neuronal, es una técnica habitual modificar las imágenes existentes para tener más ejemplos (*image augmentation*). La idea es generar imágenes ligeramente diferentes, por operaciones geométricas o ruido, pero que siguen conteniendo

claramente el mismo objeto, y por lo tanto, también nos sirven como ejemplo para la red. Estas modificaciones utilizadas son, recortar los bordes, añadir brillo y modificar la escala de la imagen de forma aleatoria.

Principales problemas en el desarrollo de este módulo. Durante el desarrollo de este módulo, se intentó utilizar la librería *Keras* en el reentrenamiento, una biblioteca que añade más opciones a la hora de realizar el reentrenamiento de una red neuronal. Una de las opciones que se intentó aprovechar con el uso de *Keras* era el *data augmentation*, que genera más imágenes para realizar el entrenamiento, a partir de las imágenes ya existentes añadiendo ciertos efectos a estas (blur, rotaciones, recortes...). Pese a que el *script* de re-entrenamiento de *Tensor Flow* también aportaba algunas de esta opciones, *Keras* aportaba más variedad y control sobre las diferentes opciones. Aunque se consiguió entrenar correctamente un modelo utilizando *Keras*, tubo que ser descartado debido a que este modelo no se podía ejecutar de forma correcta en la aplicación móvil final, y finalmente se utilizo un modelo entrenado mediante el *script* de reentrenamiento de *TensorFlow*.

Otro de los principales problemas encontrados en el desarrollo de este módulo, fue el uso de un modelo reentrenado de *TensorFlow* en un sistema operativo *Android*, debido a la gran variabilidad existente entre versiones a la hora de crear modelos reentrenados, y la falta de documentación sobre estos, y los cambios entre las diferentes versiones.

4.2. Evaluación de la CNN entrenada

Configuración del experimento. A la hora de evaluar el correcto funcionamiento de la red neuronal re-entrenada, se ha medido la eficacia de diferentes opciones de re-entrenamiento. La calidad de un modelo la medimos utilizando las siguientes **métricas**:

- Precisión (Accuracy): La precisión de un sistema clasificador, se mide comprobando para las diferentes entradas, cuantas de ellas son etiquetadas como la clase a la que realmente pertenecen. Se busca ante todo conseguir la mejor precisión posible para el sistema creado.
- Cross Entropy Loss (CE): Esta medida es muy utilizada en sistemas de clasificación cuya salida tiene una probabilidad entre cero y uno. Esta medida indica cuanto se ha distanciado la probabilidad esperada, con la generada para la salida del clasificador. Esta medida es clave durante el entrenamiento, durante el cual se busca minimizar su valor, siendo cero su valor en un sistema perfecto.

Como es habitual en cualquier entrenamiento de un clasificador, los **datos** se separan en dos conjuntos, *Training* y *Validation*, que son generados en cada entrenamiento por el *script* de reentrenamiento, seleccionando imágenes aleatorias pertenecientes a todas las imágenes etiquetadas de las diferentes clases.

Las **variaciones de entrenamiento consideradas** se han basado en modificar tres valores principalmente:

- *Num. Its.*: El número de pasos de entrenamiento realizados. Este valor es necesario ajustarlo debido a que un número muy pequeño de pasos genera

un modelo poco preciso, pero con una cantidad de pasos demasiado alta el modelo se puede sobre ajustar a los datos de entrenamiento y no funcionar correctamente con otros datos. Se probaron desde valores pequeños como 400 pasos, donde la precisión del sistema aún es baja, hasta un número de pasos muy grande como es 50.000, donde se ve que los ajustes realizados en cada iteración no mejoran los resultados del sistema.

- *Augmentation*: Variaciones como recortar márgenes de las imágenes, rotar las o añadir brillos de forma aleatoria para añadir posibles efectos que se den al utilizar el sistema.
- Arquitectura: Comparación entre las dos arquitecturas explicadas anteriormente.

Para poder medir de forma correcta la eficacia del sistema, se han ejecutado diez entrenamientos para valorar los cambios de cada opción. Esto es debido a que los conjuntos de datos de entrenamiento y validación se generan de forma aleatoria en cada entrenamiento. Al ser conjuntos de datos generados en cada entrenamiento, es posible que condicionen los resultados del entrenamiento, por lo que se tomó la decisión de promediar los resultados de varios entrenamientos para así obtener unos valores más fiables de las métricas tomadas.

Resultados del experimento. Como conclusión de esta evaluación, se decidió utilizar la arquitectura *Inception* frente a *MobileNet*, debido a que su precisión es mayor. Además se fijó el valor para el número de pasos en 5000, ya que a partir de este número de pasos, el sistema ya no mejoraba. Además, se descartó el uso de *augmentation*, ya que los resultados obtenidos con estos efectos empeoraban o no mejoraban con respecto a sus versiones sin ser utilizados, como se puede observar en la Tabla 4.4. Esto puede ser debido a utilizar una muestra de datos pequeña (60-70 imágenes por clase).

Sin embargo se puede apreciar como el sistema, en función del número de pasos de entrenamiento, ya no se produce ninguna mejoría en el sistema, al utilizar un número de pasos muy elevado, como son 10000 o más. Esto es debido a que la red ya se ha ajustado a los datos de entrenamiento, y con más pasos de entrenamiento no se mejoran los resultados, ya que ya están completamente ajustados a los datos de entrenamiento.

Cabe destacar, que todas las mediciones realizadas se basan únicamente en la primera salida que da el clasificador, pero en el sistema final se utilizaran las tres más probables. Esto hace que el sistema final implementado tenga una mayor robustez que la que se puede apreciar en estas pruebas.

Estos datos han sido obtenidos mediante el uso de *TensorBoard* [14], una herramienta de monitorización dentro de *TensorFlow*, la cuál permite visualizar información sobre el entrenamiento de redes neuronales, y a su vez obtener una gran cantidad de información sobre el grafo generado, para su posterior carga en la aplicación. Para la creación de las tablas de esta sección, se han utilizado puntos concretos de los gráficos extraídos de *TensorBoard.*[14].

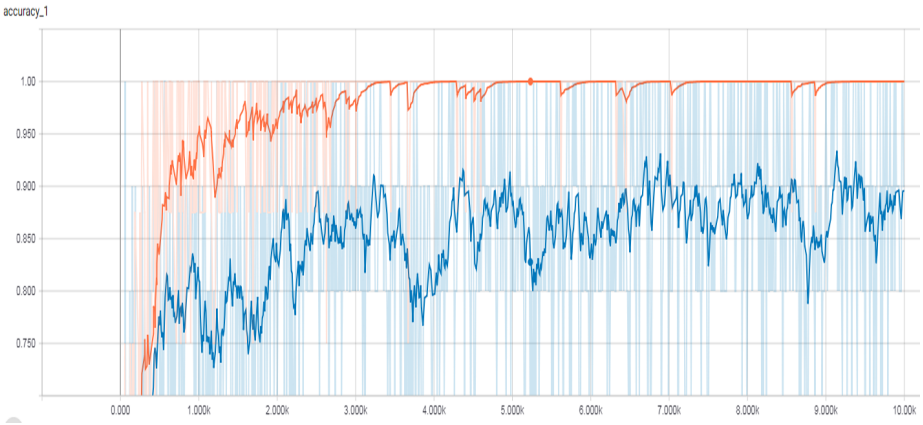


Figura 4.2: Gráfica generada mediante *TensorBoard*, para monitorizar la precisión durante un reentrenamiento del modelo utilizando la arquitectura *Inception*. Los valores naranjas corresponden a la precisión con los datos de *training*, mientras que los valores azules pertenecen a los datos de validación.

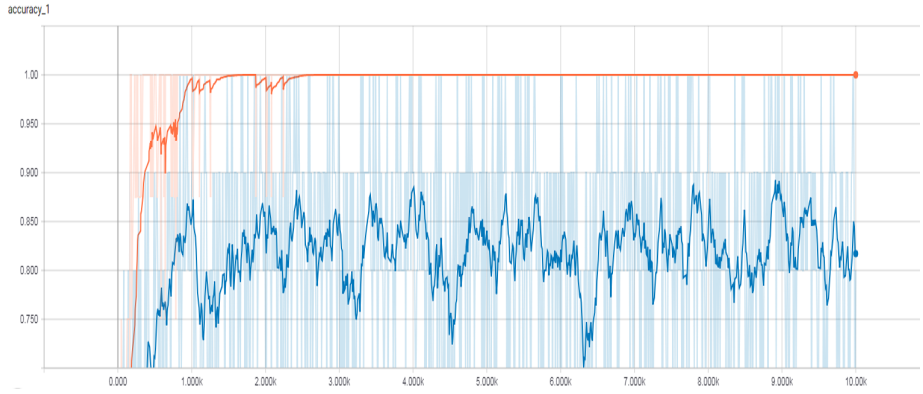


Figura 4.3: Gráfica generada de forma similar a la de la Figura 4.2, utilizando la arquitectura de *MobileNet*. Se puede apreciar que la precisión del sistema en el conjunto de validación es menor que utilizando la arquitectura *Inception*.

Num. Its.	*Training Accuracy*	*Validation Accuracy*	*Training CE*	*Validation CE*
400	0.86	0.77	0.87	1.05
5000	0.99	0.87	0.11	0.52
10000	1	0.87	0.05	0.48

Tabla 4.2: Resultados obtenidos utilizando la arquitectura *Inception*. En la primera columna se muestra el número de pasos de cada entrenamiento, y a continuación los resultados de precisión y CE. Esta arquitectura consigue alcanzar una mayor precisión por lo que ha sido elegido frente a *MobileNet* para la construcción del prototipo.

Num. Its.	*Training Accuracy*	*Validation Accuracy*	*Training CE*	*Validation CE*
400	0.92	0.78	0.72	0.96
5000	1	0.81	0.04	0.54
10000	1	0.81	0.02	0.49

Tabla 4.3: Resultados obtenidos utilizando la arquitectura *MobileNet*. Esta tabla sigue la misma distribución que la Tabla 4.2, y se puede observar como los valores de precisión obtenidos utilizando *MobileNet* son menores que utilizando la arquitectura *Inception*.

Parámetros Training		*Training*	*Validation*	*Training*	*Validation*
Num. Its.	Augment.	*Accuracy*	*Accuracy*	*CE*	*CE*
400	*Flip*	0.86	0.74	0.91	1.1
400	*Brillo*	0.83	0.75	0.93	1.1
400	*Recorte*	0.85	0.75	0.88	1.01
400	Ninguna	0.86	0.77	0.87	1.056
5000	Ninguna	0.99	0.87	0.11	0.52
50000	Ninguna	1	0.87	0.11	0.51

Tabla 4.4: Resultados obtenidos tras probar el *data augmentation* con la arquitectura *Inception*. Los valores de las 2 primeras columnas, indican las variaciones realizadas en el entrenamiento (n. de iteraciones del entrenamiento y si se ha realizado algún tipo de *data augmentation*.

4.3. Integración de la CNN en la aplicación desarrollada:

Una vez con el modelo ya entrenado, se ha procedido a cargarlo en el módulo de clasificación, reemplazando el archivo correspondiente al modelo por defecto de *TensorFlow* con el fichero del modelo reentrenado. De esta forma el clasificador recibe una imagen de la cámara, la convierte al formato y resolución adecuados, y se introduce al modelo la imagen modificada. Con la salida del modelo se determina a que objetos se aproxima más el objeto reconocido en la imagen tomada por la cámara. El clasificador no devuelve solo las clases a la que cree que pertenece la imagen a clasificar, sino que también devuelve un porcentaje que representa hasta que punto el clasificador cree que la imagen pertenece a cada clase predicha. En los resultados devueltos por el clasificador se encuentran las tres clases con las que mayor similitud tiene la imagen, aunque puede devolver un número menor, ya que se ha fijado un umbral de similitud, de tal forma que si el porcentaje de similitud para una clase es demasiado bajo, esta queda descartada directamente de los resultados. En este caso, el límite colocado ha sido de un diez por ciento.

Cabe destacar que el número de resultados devueltos por el clasificador puede ser mayor de tres, pero debido a que posteriormente, estos resultados se utilizaran para determinar contra que objetos de referencia se utilizará el sistema de reconocimiento implementado mediante *OpenCV*, tres objetos es el máximo número soportable sin que el rendimiento se vea resentido demasiado.

Además, si el objeto a detectar se encuentra centrado en la imagen, la clase a la que pertenece siempre suele encontrarse la primera o entre las tres clases con mayor probabilidad. Esto le da al sistema una mayor robustez, ya que pequeños cambios o un resultado anómalo podrían arruinar el funcionamiento de todo el sistema si solo nos quedáramos con el mejor resultado devuelto por el clasificador.

Capítulo 5

Módulo de reconocimiento de objetos basado en *local features*

Este capítulo detalla el módulo desarrollado para el reconocimiento de objetos utilizando técnicas de visión por computador, y evalúa los principales elementos de este. La función de este módulo es la de localizar en la imagen, uno de los tres objetos que el módulo descrito en el capítulo 4 le proporciona como posibles candidatos. Para localizarlo se realizaran emparejamientos entre las imágenes tomadas por la cámara, con las imágenes de referencia de los diferentes objetos almacenadas.

5.1. Localización de objetos basado en *local features*

Adquisición de descriptores de la imagen: Para poder comparar dos imágenes utilizando *local features*, el primer paso consiste en extraer los puntos de interés y los descriptores de cada una de las imágenes a comparar. Para esto, existen diferentes tipos de detectores de puntos y descriptores, y dado que en este proyecto uno de los puntos clave es el procesamiento de imágenes en tiempo real, se han comparado dos tipos de descriptores binarios:

- ORB (*Oriented Fast and Rotated BRIEF*)
- BRISK (*Binary Robust Invariant Scalable Keypoints*)

Se eligieron estos dos algoritmos, debido a que utilizaban descriptores binarios, y esto hace que los tiempos de ejecución sean mucho menores que utilizando otros algoritmos que usan otro tipo de descriptores, como podrían ser *SURF* o *SIFT*. Aunque estos dos últimos algoritmos obtienen resultados mucho más precisos, no son viables para ser utilizados en tiempo real en una aplicación móvil.

Estos puntos de interés y sus descriptores, se extraen de un conjunto de imágenes de referencia de cada objeto, y de las imágenes captadas por la cámara del móvil. Una vez extraídos estos puntos de interés y sus descriptores, se procede

a emparejar la imagen captada por la cámara del móvil con una de las imágenes pertenecientes a el conjunto de imágenes de referencia de un objeto, usando los puntos de referencia extraídos anteriormente.

Emparejamiento de imágenes: Debido a que algunos de los emparejamientos iniciales entre puntos de interés pueden ser erróneos, se deben hacer ciertos cálculos para separar los emparejamientos erróneos de los correctos. Para determinar si un emparejamiento entre dos puntos es correcto, se procede a calcular para cada punto de interés de la imagen capturada por la cámara, los dos mejores emparejamiento con la imagen del conjunto de referencia a comparar. Sólo se considerará un emparejamiento bueno cuando este sea considerablemente mejor que su segundo mejor emparejamiento, como se describe en el capítulo 2. De esta forma nos aseguramos de que, el punto de interés que vamos a emparejar, se parece mucho solo a otro punto de interés de la otra imagen, y se eliminan la mayoría de los malos emparejamientos. Los resultados de estos emparejamientos se pueden ver en la Figura 5.1.

Este proceso, se realiza para emparejar una imagen tomada por la cámara del dispositivo, con un conjunto de imágenes de referencia, para determinar cuál de los objetos que aparecen en las imágenes de referencia, se encuentra en la imagen tomada por la cámara. La imagen que obtenga un mayor número de emparejamientos correctos será la que se considerará de referencia.

Para almacenar las imágenes de referencia, se ha creado una pequeña base de datos, que contiene tres imágenes de cada objeto de referencia: una de la vista frontal, otra de la vista lateral izquierda y por último otra de la vista lateral derecha. Para más información sobre estas imágenes, revisar el anexo B.

Cabe destacar, que para reducir el tiempo de procesamiento de las imágenes tomadas por la cámara, y para evitar que se emparejaran puntos pertenecientes a objetos adyacentes al objeto que se desea reconocer, se tomó la decisión de recortar la zona centra de la imagen, y calcular los emparejamientos entre imágenes utilizando únicamente esta zona recortada.

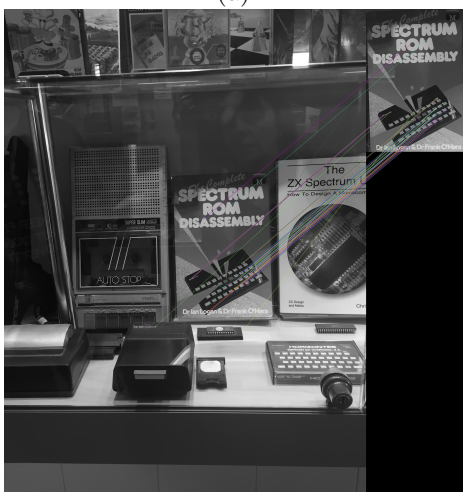
5.2. Uso de los emparejamientos para localizar un objeto

Una vez calculados los emparejamientos entre los puntos de referencia de dos imágenes, se realiza una segunda fase para eliminar cualquier emparejamiento erróneo restante, donde se hace una verificación geométrica utilizando restricciones conocidas entre las dos imágenes, que en este caso vienen dadas por los emparejamientos entre los puntos de interés calculados anteriormente.

Para realizar esta verificación geométrica, se probaron dos técnicas diferentes:

- Homografía: Cálculo de una matriz $3x3$, que proporciona las correspondencias entre puntos pertenecientes a un plano, en dos imágenes diferentes.
- Matriz Fundamental: Cálculo de una matriz $3x3$ que como la homografía, describe correspondencias entre puntos de dos imágenes, pero en este caso tiene un propósito más general, ya que no tienen por que ser puntos pertenecientes a un mismo plano (3D).

(a)



(b)

Figura 5.1: Puntos característicos emparejados entre una imagen actual de una vitrina y dos imágenes de referencia. (a) emparejamientos obtenidos usando BRISK; (b) emparejamientos obtenidos usando ORB

Figura 5.2: Puntos característicos emparejados (entre foto actual y foto del modelo) de manera robusta utilizando la matriz fundamental y ORB. Se encuadran todos los *inliers*, obteniendo así la posición donde se encuentra la zona central del objeto.

En este caso, dado que muchos de los objetos a reconocer son objetos tridimensionales, y no planos, se **decidió utilizar el cálculo de la matriz fundamental**, en lugar de utilizar el cálculo de la homografía.

Tanto utilizando la matriz de homografía como utilizando la matriz fundamental, el propósito de estas es el de verificar cuales de los emparejamientos realizados, cumplen ciertas restricciones geométricas, para así determinar que los puntos emparejados pertenecen al objeto a reconocer en la imagen. Los puntos que cumplen las restricciones geométricas entre las dos imágenes a comparar, se denominan *inliers*, y son los utilizados para determinar donde se encuentra el objeto a reconocer. Los puntos emparejados que no cumplan las restricciones geométricas son denominados *outliers*, y se descartan. Finalmente, para localizar una sección de la imagen en la que el objeto se encuentre con seguridad, se recuadran todos los *inliers*, creando así un cuadrilátero en la imagen, donde se encontrará el objeto a reconocer, como se puede ver en la Figura 5.2.

También cabe destacar que se intentó llevar a cabo una aproximación basada en determinar una serie de rectángulos predefinidos en las imágenes de referencia, para luego buscar las esquinas de dichos rectángulos en la imagen a reconocer, pero debido a la variabilidad de los puntos ORB, se tuvo que descartar ya que el resultado era muy impreciso.

5.3. Evaluación del módulo

Para evaluar este módulo, se han realizado comparaciones entre los diferentes tipos de puntos característicos, BRISK y ORB, midiendo los tiempos de cada uno de los algoritmos, y el número de emparejamientos correctos generados, considerando que se ha detectado una imagen de forma correcta cuando el número de emparejamientos correcto es igual o superior a trece. Se fijo este valor como umbral ya que el número de emparejamientos no supera este umbral cuándo se intentan emparejar dos imágenes que no contienen similitudes. Se ha buscado con estas pruebas encontrar un tipo de puntos característicos capaz de ejecutarse en tiempo real en móviles, pero que a su vez fuera lo suficientemente robusto para emparejar correctamente las imágenes con cierto grado de variación con respecto a las imágenes de referencia.

EL objetivo de esta prueba es comprobar la precisión y el tiempo de ejecución de comparaciones de imágenes utilizando puntos característicos de tipo BRISK y de tipo ORB. Este banco de pruebas también se ha utilizado para comparar con cuantas imágenes de referencia era viable comparar de forma simultánea. Para realizar dichas comparaciones, se ha seleccionado un conjunto de 25 imágenes de test y otro de cinco imágenes de referencia, siendo las imágenes de test imágenes reales de las vitrinas del museo, y las imágenes de referencia imágenes recortadas en las cuales solo aparece el objeto en concreto (ZXSpectrum2, Atari800XL, ZXSpectrum ULA, ZXSpectrum Disassembly y Commodore 64). Se ha comprobado la efectividad de los diferentes tipos de puntos y descriptores (ORB y BRISK) comprobando el número de emparejamientos correctos, entre el conjunto de imágenes de test, y el de referencia, entendiendo por un *match* correcto, aquellos que logran superar la fase de comparación de *nearest neighbour*. Los resultados completos de estas pruebas se pueden ver en el anexo A.

La Tabla 5.1 muestra un resumen de las pruebas realizadas para comparar los algoritmos de ORB y BRISK. En ella, aparecen los resultados, utilizando una, tres y cinco imágenes de referencia con las que comparar cada una de las 25 imágenes de test. Para cada una de estas tres variaciones, se muestran los datos correspondientes al tiempo promedio de procesar una imagen, y el número total de imágenes reconocidas de forma correcta.

Nº Img. Ref.	T. ORB	T. BRISK	Prec. ORB	Prec BRISK
1	6,04 segundos	11,76 segundos	23/25	25/25
3	13,16 segundos	43,4 segundos	20/25	22/25
5	19,5 segundos	66,3 segundos	16/25	18/25

Tabla 5.1: Resultados de las comparaciones entre BRISK y ORB. En la primera columna se indica el número de imágenes de referencia, en los columnas siguientes lo tiempos de ejecución para ORB y BRISK respectivamente, y las dos últimas columnas representan la precisión del sistema, en función del número de imágenes reconocidas de forma correcta.

Como **conclusión** de este experimento, se ha decidido utilizar los puntos característicos de tipo ORB, dado que aunque la precisión de los puntos BRISK es mejor, los tiempos de ejecución utilizando BRISK son demasiado altos de cara a comparar varias (tres o más) imágenes de forma simultánea, mientras

que utilizando puntos de tipo ORB se reduce mucho el tiempo de cálculo, y los resultados son similares, aunque BRISK por norma general suele generar más puntos correctos y de mayor estabilidad, ya que los puntos ORB al tener una componente aleatoria pueden variar su posición y valores. Además también se ha determinado, que tres imágenes de referencia es un número aceptable para realizar comparaciones

Capítulo 6

Aplicación móvil para el museo

Este capítulo detalla el prototipo implementado del sistema descrito en los capítulos anteriores, así como todas las decisiones de diseño tomadas durante la implementación de dicho prototipo.

6.1. Aplicación desarrollada

El prototipo implementado es una aplicación para el sistema operativo *Android*. El prototipo es capaz de identificar diferentes objetos del museo de informática [15] del edificio Ada Byron, (en Escuela de Ingeniería y Arquitectura, ubicada en el campus Río Ebro, de la Universidad de Zaragoza), y una vez identificado el objeto, añadir información aumentada en tiempo real en la visualización de dicha imagen en el móvil.

6.1.1. Base de datos y modelo de la aplicación final

Para la versión final del prototipo construido, existen tres conjuntos de datos utilizados por lo módulos anteriormente descritos:

1. Base de datos de imágenes utilizadas para reentrenar CNN. Este conjunto de datos es descrito en mayor detalle en el capítulo 4.

2. Base de datos de imágenes de referencia utilizadas en el módulo de reconocimiento basado en *features*. Por cada objeto a reconocer se almacenan tres imágenes de referencia. Más información sobre este conjunto de datos en el anexo B.

3. Base de datos de información relativa a los objetos, compuesta por un texto breve para todos los objetos y algunas imágenes (tres por objeto) para una visualización adicional en algunos objetos, como se ve en la Figura 6.2 (b).

Los objetos para los que se han recopilado datos y funciona actualmente el prototipo son 15: AmstradCPC 464, Apple 2, Atari 520ST, Atari 800XL,

Commodore 64, Commodore Amiga, IBM PorwerPC 601, Macintosh ED, Radio ShackTRS, Sinclair QL, The ZXSpectrum ULA (libro), Toshiba 64k, ZXSpectrum, ZXSpectrum2 y ZXSpectrum 81. No se han añadido más objetos debido a la gran cantidad de imágenes necesarias por cada objeto nuevo a añadir.

6.1.2. Funcionamiento del sistema

Las **fases principales** de funcionamiento una vez que arranca el programa son las siguientes:

1. **Iniciar el reconocimiento, utilizando la CNN** descrita en el capítulo 4, para obtener tres candidatos, de entre los posibles objetos de la base de datos, que son los más probables de aparecer en el imagen actual. Esta fase lleva un tiempo aproximado de dos segundos por cada fotograma procesado por la CNN, por lo que cada vez que se utiliza la CNN en el prototipo la pantalla queda momentáneamente congelada. Para evitar congelar la pantalla de forma continua, la CNN se utiliza a intervalos de 50 fotogramas cuando no hay ningún candidato reconocido.

2. De los **tres candidatos obtenidos con la CNN, se refina la búsqueda del objeto que aparece realmente en la imagen, mediante emparejamientos y verificaciones geométricas**, para verificar que objeto y donde está situado en la imagen. Inicialmente la comparación se realiza con las tres imágenes almacenadas de los tres candidatos, lo que conlleva un tiempo de procesado de aproximadamente dos segundos por cada fotograma, y una vez se ha localizado el objeto, en los siguientes fotogramas se comparará con la imagen encontrada, lo que hace que estos fotogramas puedan ser procesados en milisegundos. En caso de perder la última imagen localizada, se comparará con las otras dos imágenes de referencia del objeto, lo que hace que procesar un fotograma sea levemente más costoso que el caso anterior, pero sin llegar a costar un segundo.

3. Una vez encontrado el objeto que aparece en la imagen y donde se encuentra, **se procede a emparejar, de forma constante, cada imagen que llega desde la cámara del dispositivo con las imágenes de referencia del candidato seleccionado.**

4. Finalmente, **se añade la información correspondiente al objeto reconocido**, en la posición donde se ha detectado el objeto. En el momento en el que se deje de detectar el objeto, el sistema volverá al estado inicial.

De estas fases principales de ejecución de la aplicación, podemos distinguir las que son de procesado de la imagen para reconocimiento, y las que son de visualización.

En cuanto a las de **procesado de la imagen**, están representadas en la máquina de estados de la Figura 6.1 Como se puede apreciar, el estado inicial del sistema es cuando aún no ha detectado ningún objeto. Una vez iniciado el reconocimiento, se pueden distinguir las tres fases principales del sistema, detalladas a continuación.

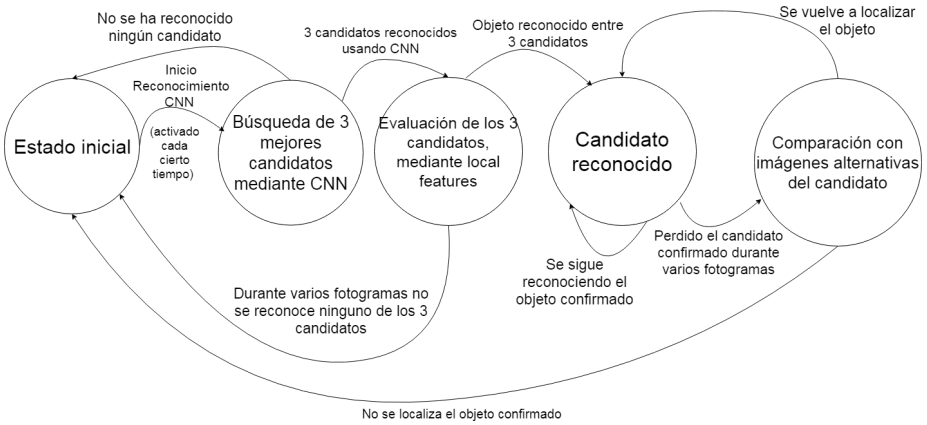


Figura 6.1: Diagrama de estados de la ejecución de las fases de procesamiento de imagen para reconocimiento de objetos automático.

Estado: búsqueda candidatos mediante CNN. El primer estado cuando el sistema sale de reposo, consiste en tomar dos imágenes mediante el dispositivo *Android*, y proceder a clasificarlas mediante la CNN implementada en el módulo de reconocimiento inicial.

Los resultados de clasificación de estas dos imágenes se promedian, para así tener un funcionamiento más robusto del sistema frente una imagen mal clasificada. La forma de ponderar las medias de los candidatos, es sumando un voto por cada vez que se detecte un candidato, como se ve en la ecuación 6.1. En esta ecuación, n es el número de clases, y j toma valores entre uno y dos, ya que se promedian dos fotogramas. La función $aparece(i, j)$ indica si el candidato i ha sido reconocido en la imagen j.

$$\bigvee i, (0 \leq i < n) \Rightarrow N_i = \sum_{j=1}^{2} aparece(i, j) \tag{6.1}$$

Finalmente, se seleccionan los tres candidatos con valor N_i más alto.

No se utilizan los porcentajes de fiabilidad generados también por el clasificador, debido a que un resultado espúreo, puede obtener un porcentaje anormalmente alto, y por tanto aparecer por encima de otros candidatos, propiciando así que el candidato correcto no se encuentre entre los tres seleccionados finalmente.

Estado: Evaluación de candidatos mediante *local features*. Una vez se ha realizado la selección de los tres mejores candidatos en el estado anterior, para confirmar que objeto aparece en la imagen, se realiza, durante 50 fotogramas, comparaciones contra las tres imágenes de referencia almacenadas para cada uno de los tres posibles objetos. Es decir, se hacen comparaciones mediante las características locales elegidas (en este caso se seleccionarón las características de tipo ORB) con un máximo de nueve imágenes. Cuando el resultado de esta comparación supera cierto umbral de *matches* encontrados (entre la imagen de la cámara y una de las nueve imágenes de referencia), se considera que el objeto ha sido encontrado, y se finaliza esta fase de comparaciones iniciales.

Con uno de los tres candidatos ya confirmado, las siguientes iteraciones de comparaciones se realizaran solo con las tres imágenes de el candidato confirmado. Además, se guarda cuál de las tres imágenes de referencia ha sido la que se ha emparejado correctamente la última vez, y es la imagen que se utilizará para hacer las comparaciones con las siguientes imágenes obtenidas de la cámara del dispositivo.

En el caso de que en las siguientes imágenes obtenidas, no se encuentren correspondencias suficientes con la última imagen de referencia emparejada, se procede a comparar con las otras dos imágenes de referencia del candidato confirmado. Y en caso de que no se detecte durante un periodo de tiempo ninguna de las tres imágenes del candidato confirmado, se procede a limpiar la lista de candidatos y a volver al estado inicial del sistema.

Además cabe destacar, que el emparejamiento de imágenes no se realiza en todos los fotogramas capturados por la cámara, sino que se realiza cada pequeños intervalos, y en los fotogramas donde no se ha realizado el emparejamiento, se utiliza como posición del objeto en la imagen para la visualización, la última posición reconocida del objeto.

Visualización: Dado que independientemente del estado del sistema, la aplicación debe mostrar algo por pantalla, los estados de visualización del sistema se ejecutan de forma paralela al resto de estados del sistema. Estos estados, dependen fundamentalmente de si se ha detectado un objeto, o si el sistema se encuentra aún buscando objetos. Estas dos **fases de visualización** del sistema, consisten básicamente en dos estados, que se pueden ver en la Figura 6.3 Estos dos estados son:

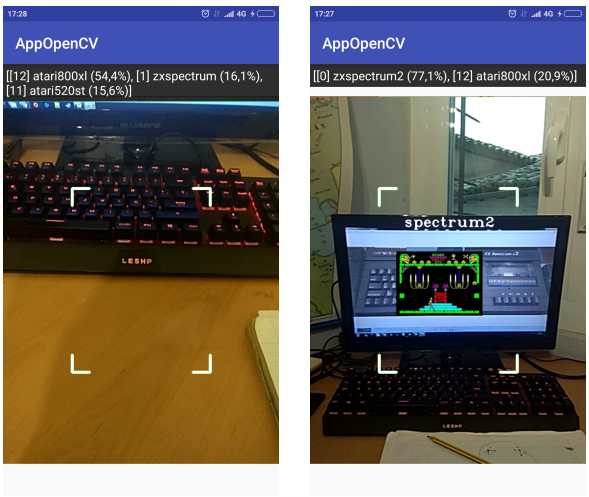
- Si objeto no localizado: mostrar delimitadores de búsqueda.
- Si objeto localizado: mostrar información relacionada con el objeto.

6.2. Pruebas de integración de los módulos

6.2.1. Pruebas realizadas

Para comprobar el correcto funcionamiento del prototipo construido, se crearon dos aplicaciones diferentes, la primera orientada a la depuración del sistema implementado, y otra ya de cara a la visualización final.

- **La aplicación de depuración** como se puede apreciar en Figura 6.4(a), no añade ninguna información sobre el objeto detectado a parte de su nombre, pero muestra la sección en la cuál están comprendidos los puntos emparejados de forma correcta. De esta forma, se puede comprobar en directo con los objetos reales del museo si el funcionamiento de los emparejamientos es correcto, o por el contrario se están emparejando puntos no pertenecientes al objeto a reconocer. Además añade un botón para poder iniciar el reconocimiento del objeto en el momento deseado, para así poder depurar con mayor facilidad la aplicación en cada uno de los diferentes objetos.

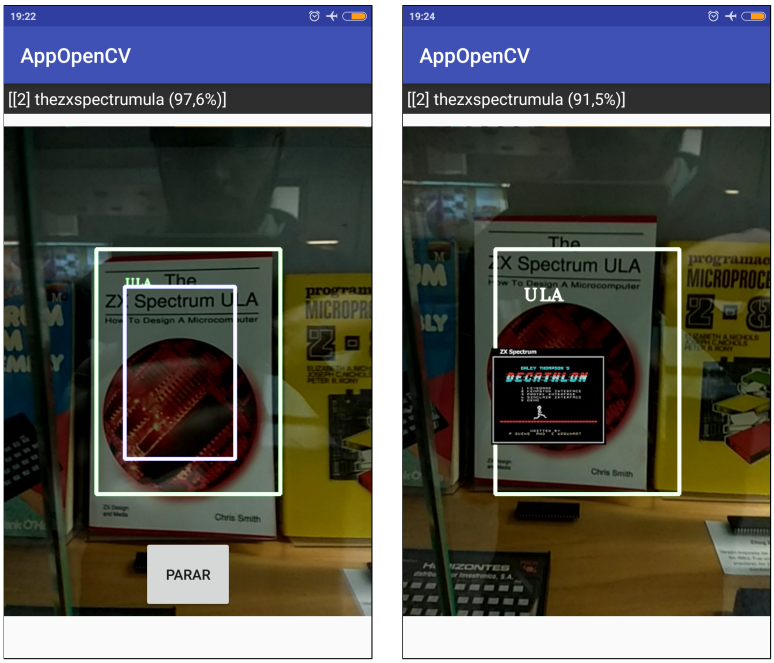


(a) (b)

Figura 6.2: Imágenes pertenecientes a la aplicación final. (a): Aún no se ha reconocido ningún objeto y se muestran lo delimitadores de la zona a reconocer. (b): Objeto localizado en la imagen. Se puede apreciar como se proyecta sobre el objeto su nombre y una imagen relacionada con el objeto detectado.

Figura 6.3: Diagrama de estados detallado de las fases de visualización del sistema.

- **La aplicación final** contiene una visualización más depurada, en la cuál se muestran las imágenes asociadas al objeto reconocido en el centro del área donde ha sido detectado el objeto. En esta versión se ha eliminado el marco que señala la zona donde se encuentran los emparejamientos correctos, y el botón que inicia el reconocimiento, haciendo que el reconocimiento se inicie de forma automática en intervalos de tiempo, y que a su vez, cuando un objeto no haya sido detectado en un periodo de tiempo, se descarte el objeto y se vuelvan a buscar nuevos candidatos. Este prototipo



(a) (b)

Figura 6.4: Captura de pantalla de la ejecución de las dos aplicaciones implementadas. (a) Aplicación de depuración. En ella se pueden observar dos rectángulos. El exterior es la zona de detección de objetos, y el interior muestra la zona en la que se ha detectado el objeto. (b) Aplicación final con visualización de información adicional.

se corresponde con la Figura 6.4 (b).

Las pruebas realizadas con estas aplicaciones han sido:

- Reconocimiento de los quince objetos cargados a diferentes horas del día, para comprobar hasta que punto puede afectar la variación de la iluminación en la detección de los objetos.
- Comprobación del correcto funcionamiento de la aplicación con objetos de vitrinas centrales, al cambiar el punto de vista desde el que son vistos (de frente, lateral izquierdo o lateral derecho).
- Pruebas realizadas apuntando a otros objetos para comprobar que el sistema no confunde objetos a reconocer con otros fuera del sistema.

6.2.2. Discusión

Esta subsección presenta una breve discusión de los principales puntos fuertes y débiles de la versión actual del prototipo.

Escalabilidad del sistema: Uno de los puntos fuertes de el sistema propuesto es su escalabilidad de cara a añadir objetos nuevos.

El primer módulo implementado que se ejecuta, es la red neuronal convolucional, que analizará la imagen completa, y seleccionará los tres objetos de referencia más similares. El tiempo de ejecución de este proceso es independiente del número de clases que tenga la red, por lo que a la hora de ejecutar el sistema propuesto con un número mayor de objetos a reconocer, el tiempo de ejecución sería el mismo que con menos objetos.

De esta forma, aunque la colección de objetos a reconocer aumentara y se modificara la red neuronal para que funcionara con más clases, tampoco afectaría a el módulo de visión por computador implementado, ya que el número de imágenes contra las que comparar seguirá siendo siempre como máximo nueve imágenes.

Aunque cabe destacar, que para poder aumentar el número de clases de la red neuronal convolucional, es necesario recopilar un conjunto de imágenes diferentes del objeto a añadir, y luego re entrenar la red neuronal con este nuevo conjunto de imágenes. Se puede ver una descripción de como añadir nuevas clases en el anexo C.

Visualización de la Realidad Aumentada: Aunque finalmente se decidió implementar el preprocesado de candidatos con *TensorFlow*, y de la librería *OpenCV* de cara a hacer una sencilla visualización, también se barajó el uso de la librería *ARCore* [16], que es capaz de reconocer planos y proyectar sobre ellos modelos en tres dimensiones, y hacer un seguimiento de esos puntos en tiempo real, para poder mantener el modelo de tres dimensiones actualizado en función de la posición desde la que la cámara ve esos puntos. Aunque esta aplicación tiene un gran potencial de cara a la realidad aumentada, finalmente acabo descartándose su uso, ya que al ser una librería muy novedosa y de altas prestaciones, requiere para su uso dispositivos muy nuevos y de gama alta. Este requisito es muy restrictivo, y hace que solo se pueda utilizar un reducido y selecto grupo de dispositivos *Android*, de los cuáles no se disponían durante la realización de este proyecto.

Capítulo 7

Conclusiones

7.1. Conclusiones técnicas

Los objetivos planteados para este proyecto, eran dos a grandes rasgos: crear una aplicación para dispositivos móviles capaz de reconocer y localizar objetos en tiempo real, y el visualizar información aumentada relacionada con el objeto detectado, también en tiempo real.

Ambos objetivos han sido cumplidos en el prototipo construido, pero hay que destacar que la mayoría del trabajo se ha realizado en el cumplimiento del primer objetivo.

Este primer objetivo, la localización y el reconocimiento de los objetos, se ha alcanzado mediante el uso de técnicas de visión por computador combinadas con técnicas de *deep learning*. Se realizaron varias comparaciones entre diferentes aproximaciones, y finalmente se utilizaron un sistema de reconocimiento basado en la CNN reentrenada con la arquitectura *Inception* para detectar de manera eficiente los objetos más probables, y luego un sistema más detallado de localización del objeto basado en características locales de tipo *ORB*.

En cuanto al segundo objetivo general, se ha implementado una versión sencilla de visualización debido a que la parte del reconocimiento y localización de objetos ha sido en la que más se ha profundizado, ya que su correcto funcionamiento era indispensable para el resto de la aplicación. En ella, se añade en tiempo real información adicional del objeto que se está viendo (el nombre del objeto y diferentes imágenes relacionadas con el objeto reconocido). Este segundo apartado podría mejorarse como se describe más adelante, en la sección de trabajo futuro.

El resultado final de todo el proyecto, ha sido la aplicación prototipo desarrollada para móviles *Android*, la cuál realiza de forma automática y en tiempo real los 2 objetivos citados anteriormente, utilizando los objetos almacenados en el museo del edificio Ada Byron como elementos a reconocer, y sobre los que ha sido probada.

7.2. Problemas encontrados

A continuación se van a detallar algunos de los principales problemas encontrados durante el desarrollo de este proyecto.

Uso de *Android*: Uno de los principales problemas con los que se ha lidiado durante la construcción del sistema, ha sido el trabajo con el sistema operativo *Android*. El tiempo de puesta en marcha de los entornos y la integración con *Android* llevó más tiempo del esperado inicialmente.

Dado que los conocimientos iniciales sobre el manejo de elementos del sistema de *Android* eran escasos, esto en un principio supuso un reto, debido a que antes de poder profundizar en los módulos de visión y *deep learning*, era necesario conseguir una aplicación base sobre la que trabajar, y esto planteó algunos problemas, principalmente a la hora de depurar ciertas secciones del sistema dependientes de *Android*, como son el formato y las dimensiones de las imágenes obtenidas de la cámara, la carga de imágenes y su transformación para poder ser utilizadas de forma correcta en *OpenCV* con las imágenes obtenidas de la cámara, y un largo etcétera de situaciones donde se requieren ciertos conocimientos del sistema interno de *Android*.

Para lidiar con este tipo de problemas, la estrategia planteada para depurar el módulo de reconocimiento basado en características locales, era inicialmente desarrollar el código en un ordenador con *OpenCV* y depurar el funcionamiento de la funcionalidad deseada en ese entorno controlado, y una vez que se ha comprobado el correcto funcionamiento en ese entorno, se migraba el código a la aplicación, para resolver posibles problemas derivados del manejo de *Android* con la certeza de que la funcionalidad estaba correctamente implementada, y que cualquier error que apareciera estaba relacionado con el manejo de *Android*.

Uso e integración de versiones de nuevas librerías: Otro de los principales problemas que se han encontrado a la hora de trabajar con el sistema operativo *Android*, ha sido el de la integración de diferentes librerías, como son *OpenCV* y *TensorFlow*. En el caso concreto de esta segunda, al ser una librería relativamente nueva, existen diferentes versiones no estables, que se modifican cada poco tiempo. Esto generó un problema, a la hora de reentrenar un modelo de red neuronal convolucional, que fue que muchas de las diferentes versiones de *scripts* proporcionados por *TensorFlow* para el reentrenamiento de modelos, generaban modelos no compatibles con las últimas versiones de la aplicación *Android*. Finalmente, si que se consiguió resolver este problema utilizando versiones anteriores de *TensorFlow*.

Actualmente, *TensorFlow* ha planteado una forma de resolver este problema, creando una versión reducida de su librería, denominada *TensorFlow Lite*, la cuál trae las herramientas necesarias para trabajar con esta librería en el sistema operativo *Android*, de forma más fácil y sencilla que utilizando la librería completa, pero que a su vez tienes menos opciones y se encuentra más restringida que la versión completa de la librería.

Requisitos de ARCore: Para mejorar la parte de la visualización de la aplicación, se intentó utilizar la herramienta *ARCore*, la cual aporta muchas ventajas como *tracking* automático de planos, o visualizaciones de objetos tridimensionales complejas. Pero finalmente se decidió no utilizarse por sus altos requisitos.

El problema que trae intrínseco *ARCore*, es que al ser una herramienta desarrollada muy recientemente, y con un gran potencial, requiere de dispositivos con una gran capacidad de cálculo y con las últimas versiones del sistema operativo

Android. Está fue la principal limitación a la hora de utilizar esta herramienta en el sistema presentado, ya que no se disponía de un dispositivo que cumpliera con los requisitos de esta herramienta, y aunque se consiguió probar su funcionamiento en un dispositivo simulado virtualmente, finalmente acabó desechándose de cara al prototipo final presentado.

Otro de los problemas que presentaba el uso de *ARCore*, es que se trata de una herramienta en continuo desarrollo, y que aún no cuenta con versiones muy estables, lo que favorece la aparición de *bugs*, y posibles incompatibilidades entre versiones, ya que las aplicaciones que utilicen esta herramienta, requieren tener la última versión de *ARCore* instalada.

7.3. Trabajo futuro

De cara a un posible trabajo futuro, el apartado que requiere una mayor profundización sería el de la visualización, ya que el sistema propuesto se ha orientado más a obtener una detección y localización robusta y en tiempo real.

Como se ha comentado, el prototipo construido utiliza una visualización muy sencilla, mostrando únicamente imágenes relacionadas, que se colocan en la posición donde el objeto está siendo detectado. La profundización en el uso de herramientas como *ARCore*, proporcionarían posibilidades más complejas y elaboradas a la hora de hacer una visualización de información más elaborada. Esto es debido a que esta herramienta, proporciona la capacidad de colocar objetos tridimensionales sobre planos detectados, dando lugar a visualizaciones más realistas y con más posibilidades.

Otro de los puntos que podrían resultar interesantes de cara al futuro de la aplicación, es la creación de una herramienta para facilitar el añadir nuevas clases al prototipo. Actualmente, por cada objeto añadido, se requieren tres imágenes donde se vea el objeto de forma nítida, y aproximadamente sesenta imágenes para añadir la clase a la CNN, mediante el reentrenamiento de esta. Una herramienta que recibiera estas imágenes y se encargara de forma automática de añadirlas a la aplicación facilitaría mucho este aspecto de cara a ser utilizada con más objetos o incluso en otros museos.

Además, otro posible ámbito de cara al trabajo futuro, sería el de la optimización de la aplicación, de cara a poder mejorar el número de fotogramas por segundo al que la aplicación trabaja en sus momentos de mayor carga. Esto podría realizarse mediante la implementación de un sistema de *tracking* de los puntos detectados (en el caso de no añadir *ARCore*, ya que esta herramienta hace *tracking* de planos de forma automática), para así reducir la carga de trabajo una vez ya se haya localizado el objeto que se quiere detectar.

Apéndice A

Resultados Reconocimiento Basado en *Features*

En este anexo, se detallan los resultados obtenidos durante las comparaciones de ORB y BRISK. Este anexo detalla las pruebas realizadas con un conjunto de 25 imágenes obtenidas de las vitrinas del museo, contra cinco imágenes de referencia, que contienen únicamente el objeto a reconocer. En este caso, los objetos a reconocer elegidos para las pruebas son:

- ZXSpectrum 2.
- Atari800XL.
- Commodore 64.
- ZXSpectrum ULA (libro).
- Spectrum ROM disassembly (libro).

En la Tabla A.1 se muestra que objeto u objetos aparecen en cada una de las 25 imágenes que componen el conjunto de datos.

Las siguientes tablas, muestran resultados detallados de las comparaciones entre BRISK y ORB con las imágenes anteriormente explicadas. En los resultados se puede observar para cada imagen el tiempo de cálculo, si se ha emparejado correctamente con la imagen de referencia que contiene, y el número de *matches* obtenidos. Para considerar dos imágenes correctamente emparejadas, el número de *matches* debe superar un umbral, que en este caso ha sido fijado en trece, como se explica en el capítulo 5.

Numero de imagen	Objetos que aparecen
1	ZXSpectrum ULA y Spectrum ROM disassembly
2	ZXSpectrum ULA y Spectrum ROM disassembly
3	ZXSpectrum ULA y Spectrum ROM disassembly
4	ZXSpectrum ULA y Spectrum ROM disassembly
5	ZXSpectrum ULA y Spectrum ROM disassembly
6	Atari800XL
7	Atari800XL
8	Atari800XL
9	Atari800XL
10	Atari800XL
11	Commodore 64
12	ZXSpectrum 2
13	ZXSpectrum 2
14	ZXSpectrum 2
15	ZXSpectrum 2
16	ZXSpectrum 2
17	ZXSpectrum 2
18	ZXSpectrum 2
19	Commodore 64
20	Commodore 64
21	Commodore 64
22	Commodore 64
23	Commodore 64
24	ZXSpectrum ULA y Spectrum ROM disassembly
25	Atari800XL

Tabla A.1: Correspondencias entre cada una de las 25 imágenes utilizadas en las siguientes pruebas, y los objetos a reconocer que aparecen en cada una de ellas.

Nº	Tiempo(segs)	Resultado BRISK	Nº matches
1	89	true positive	244
2	79	true positive	26
3	110	true positive	65
4	106	true positive	265
5	85	true positive	87
6	13	true positive	44
7	8	false negative	6
8	10	false negative	0
9	12	false negative	2
10	7	false negative	1
11	42	true positive	102
12	94	true positive	35
13	113	true positive	32
14	57	true positive	31
15	68	true positive	13
16	132	true positive	14
17	190	true positive	350
18	81	false negative	11
19	22	false negative	6
20	58	true positive	700
21	35	true positive	18
22	36	false negative	7
23	63	true positive	18
24	97	true positive	50
25	50	true positive	15
Medias:	66,28	18/25	85,8

Tabla A.2: Resultados obtenidos utilizando puntos característicos de tipo BRISK, comparando una imagen de test con 5 de referencia.

Nº	Tiempo(segs)	Resultado ORB	Nº matches
1	19	true positive	15
2	21	false negative	6
3	19	true positive	42
4	19	true positive	61
5	19	true positive	27
6	15	true positive	145
7	16	true postive	54
8	14	false negative	3
9	14	false negative	4
10	14	false negative	3
11	12	true positive	112
12	22	true positive	90
13	22	true positive	35
14	22	true positive	18
15	22	false negative	4
16	22	false negative	8
17	21	true positive	481
18	21	true positive	13
19	23	false positive	13
20	22	true positive	673
21	21	true positive	16
22	21	false positive	14
23	23	true positive	20
24	19	true positive	45
25	24	false negative	7
Medias:	19,5	16/25	73,3

Tabla A.3: Resultados obtenidos utilizando puntos característicos de tipo ORB, comparando una imagen de test con 5 de referencia.

Nº	Tiempo (segs)	Resultado BRISK	Nº matches
1	59	true positive	244
2	52	true positive	18
3	70	true positive	57
4	68	true positive	74
5	56	true positive	69
6	7	true negative	3
7	5	true negative	2
8	7	true negative	0
9	7	true negative	2
10	5	true negative	1
11	30	true positive	102
12	60	true positive	35
13	72	true positive	32
14	39	true positive	31
15	49	true positive	13
16	83	true positive	14
17	122	true positive	350
18	54	false negative	10
19	16	false negative	6
20	39	true positive	700
21	24	true positive	18
22	25	false negative	7
23	44	true positive	18
24	60	true positive	50
25	31	true negative	10
Medias:	43,36	22/25	73,64

Tabla A.4: Resultados obtenidos utilizando puntos característicos de tipo BRISK, comparando una imagen de test con 3 de referencia. En concreto las imágenes de referencia son de los objetos ZXSpectrum ULA (libro), Spectrum 2 y Commodore 64.

Nº	Tiempo (segs)	Resultado ORB	Nº matches
1	12	true positive	15
2	13	false negative	6
3	13	false negative	7
4	13	true positive	30
5	13	true positive	27
6	10	true negative	8
7	9	true negative	4
8	9	true negative	3
9	9	true negative	4
10	8	true negative	2
11	15	true positive	112
12	14	true positive	90
13	13	true positive	35
14	15	true positive	18
15	15	false negative	4
16	15	false negative	8
17	14	true positive	481
18	15	true positive	13
19	14	true positive	13
20	16	true positive	673
21	16	true positive	16
22	13	false positive	14
23	15	true positive	20
24	14	true positive	45
25	16	true negative	7
Medias:	13,16	20/25	66,2

Tabla A.5: Resultados obtenidos utilizando puntos característicos de tipo ORB, comparando una imagen de test con 3 de referencia. En concreto las imágenes de referencia son de los objetos ZXSpectrum ULA (libro), Spectrum 2 y Commodore 64.

Nº	Tiempo (segs)	Resultado BRISK	Nº matches
1	16	true positive	244
2	14	true positive	18
3	19	true positive	57
4	18	true positive	74
5	15	true positive	69
6	2	true negative	0
7	2	true negative	2
8	1	true negative	0
9	2	true negative	2
10	1	true negative	1
11	2	true negative	4
12	17	true negative	12
13	20	true negative	12
14	11	true negative	7
15	20	true negative	7
16	25	true negative	7
17	31	true negative	7
18	15	true negative	7
19	6	true negative	6
20	12	true negative	6
21	7	true negative	1
22	8	true negative	7
23	13	true negative	7
24	17	true positive	50
25	10	true negative	4
Medias:	11,76	25/25	24,4

Tabla A.6: Resultados obtenidos utilizando puntos característicos de tipo BRISK, comparando una imagen de test con 1 de referencia. En este caso, la imagen de referencia a comparar es ZXSpectrum ULA (libro).

Nº	Tiempo (segs)	Resultado ORB	Nº matches
1	6	true positive	15
2	6	false negative	6
3	7	false negative	7
4	6	true positive	30
5	6	true positive	27
6	3	true negative	0
7	3	true negative	3
8	3	true negative	1
9	3	true negative	4
10	3	true negative	2
11	6	true negative	1
12	6	true negative	1
13	5	true negative	5
14	6	true negative	3
15	4	true negative	4
16	8	true negative	8
17	7	true negative	5
18	7	true negative	4
19	6	true negative	3
20	6	true negative	2
21	6	true negative	3
22	7	true negative	10
23	15	true negative	4
24	8	true positive	45
25	8	true negative	5
Medias:	6,04	23/25	7,92

Tabla A.7: Resultados obtenidos utilizando puntos característicos de tipo ORB, comparando una imagen de test con 1 de referencia. En este caso, la imagen de referencia a comparar es ZXSpectrum ULA (libro).

Apéndice B

Base de datos de imágenes utilizadas.

Este anexo detalla la información referente a la base de datos de imágenes y información relacionada que se ha creado, y tiene tres partes principales: 1) Imágenes utilizadas para el re-entrenamiento de la CNN. 2) Imágenes de referencia utilizadas en el módulo de reconocimiento basado en *local features*. 3) Imágenes utilizadas para visualización aumentada.

B.1. Datos re-entrenamiento de la CNN

La Tabla B.1 detalla las imágenes utilizadas para el re-entrenamiento de la CNN utilizada en el primer módulo del sistema, utilizando quince objetos diferentes del museo.

Objeto	**Nº Imágenes Museo**	**Nº Imágenes Internet**
AmstradCPC	5	55
Apple2	14	46
Atari520ST	8	52
Atari800XL	5	55
Commodore64	7	53
Commodore Amiga	12	49
IBM PowerPC	20	42
Macintosh ED	13	47
ShackTRS	3	58
Sinclair QL	4	56
ZXSpectrum ULA	22	38
Toshiba 64k	6	54
ZXSpectrum	6	55
ZXSpectrum 2	11	49
ZXSpectrum 81	6	54

Tabla B.1: Distribución de las imágenes recopiladas para realizar los re-entrenamientos de la CNN de reconocimiento.

B.2. Imágenes de referencia del módulo de reconocimiento con *local features*

Por cada uno de los quince objetos reconocidos, se han almacenado tres imágenes. Estas tres imágenes se corresponden a la vista frontal del objeto, la vista desde el lateral derecho, y la vista desde el lateral izquierdo. En caso de que alguno de los objetos no pueda ser observado desde tres perspectivas diferentes, se completa el conjunto de tres imágenes con otras imágenes del objeto. Como se puede apreciar, algunas de las imágenes han sido editadas para eliminar las zonas en las cuales apareciera un objeto diferente al de referencia de la imagen.

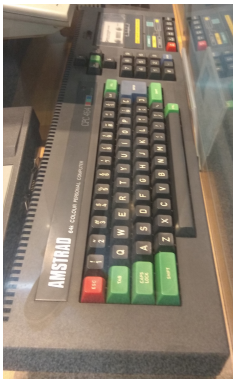
Vista Frontal | Lateral derecha | Lateral izquierda

Figura B.1: Imágenes utilizadas para el reconocimiento del objeto **Atari 800 XL**.

Vista Frontal | Lateral derecha | Lateral izquierda

Figura B.2: Imágenes utilizadas para el reconocimiento del objeto Commodore Amiga.


Vista Frontal



Lateral derecha

Lateral derecha 2

Figura B.3: Imágenes utilizadas para el reconocimiento del objeto Amstrad CPC 464.


Vista Frontal


Lateral derecha


Lateral izquierda

Figura B.4: Imágenes utilizadas para el reconocimiento del objeto Atari 520.

Vista Frontal


Lateral derecha


Lateral izquierda

Figura B.5: Imágenes utilizadas para el reconocimiento del objeto Apple 2.

Vista Frontal Lateral derecha Lateral izquierda

Figura B.6: Imágenes utilizadas para el reconocimiento del objeto Commodore 64.

Vista Frontal Lateral derecha Lateral izquierda

Figura B.7: Imágenes utilizadas para el reconocimiento del objeto IBM PowerPC 601.

Vista Frontal Lateral derecha Lateral izquierda

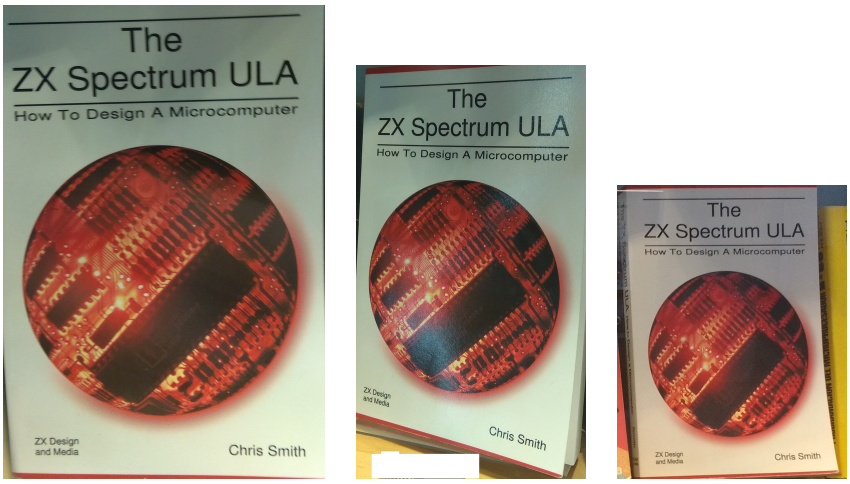
Figura B.8: Imágenes utilizadas para el reconocimiento del objeto Macintosh ED.

Vista Frontal Lateral derecha Lateral derecha completa

Figura B.9: Imágenes utilizadas para el reconocimiento del objeto Shack TRS.

Vista Frontal Lateral izquierda Lateral derecha

Figura B.10: Imágenes utilizadas para el reconocimiento del objeto Sinclair QL.



Vista Frontal Lateral derecha Lateral izquierda

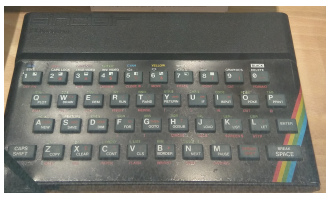
Figura B.11: Imágenes utilizadas para el reconocimiento del objeto The ZXSpectrum ULA (libro).

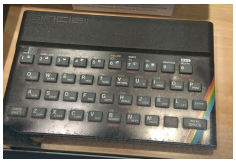

Vista Frontal


Lateral derecha


Lateral derecha completo

Figura B.12: Imágenes utilizadas para el reconocimiento del objeto Toshiba 64k.


Vista Frontal


Lateral derecha

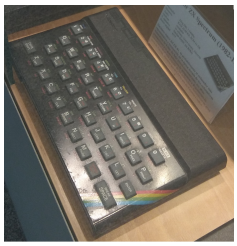
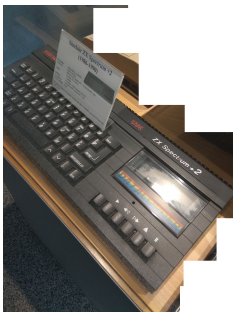

Lateral izquierda

Figura B.13: Imágenes utilizadas para el reconocimiento del objeto ZXSpectrum.


Vista Frontal


Lateral derecha

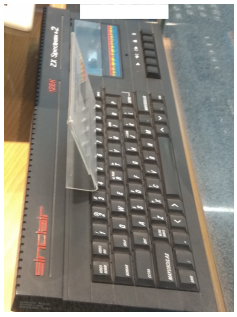

Lateral izquierda

Figura B.14: Imágenes utilizadas para el reconocimiento del objeto ZXSpectrum 2.


Vista Frontal


Lateral derecha


Lateral izquierda

Figura B.15: Imágenes utilizadas para el reconocimiento del objeto ZXSpectrum 81.

B.3. Imágenes para visualización aumentada

Para comprobar el correcto funcionamiento de la visualización aumentada implementada, se recopilaron tres imágenes relacionadas con tres objetos diferentes (ZXSpectrum ULA(libro), ZXSpectrum 2 y Commodore 64).

Figura B.16: Imágenes utilizadas para la visualización aumentada del objeto ZXSpectrum 2.

Figura B.17: Imágenes utilizadas para la visualización aumentada del objeto ZXSpectrum ULA (libro).

Figura B.18: Imágenes utilizadas para la visualización aumentada del objeto Commodore 64.

Apéndice C

Manual para introducir nuevos objetos.

Este anexo detalla los pasos a seguir para poder añadir nuevos objetos a reconocer en la aplicación implementada. Este proceso, es relativamente sencillo y es uno de los puntos fuertes de esta aplicación, ya que el uso de CNNs la hace altamente escalable, y lo único que se debe realizar es añadir nuevas clases a la CNN re-entrenada, y añadir las imágenes de referencia del objeto para luego poder se reconocido por el módulo de reconocimiento basado en *local features*.

Los pasos a seguir para añadir un nuevo objeto son los siguientes:

1. Recolectar entre 60 y 70 imágenes del objeto. Para la recolección de estas imágenes es fundamental que sean de diferentes fuentes, o que al menos el objeto aparezca con fondos diferentes en la mayoría de ellas, ya que sino la CNN confundirá el fondo de la imagen con el objeto. En este caso, se recomienda utilizar unas cuantas imágenes obtenidas de los objetos que serán reconocidos de forma final, y el resto obtenerlas de Internet para una mayor variación de iluminaciones, sombras, fondos, ...

2. Introducir estas imágenes en una carpeta con el nombre del objeto a reconocer, y añadir la carpeta al directorio donde están almacenadas las carpetas que contienen el resto de los objetos.

3. Re-entrenar la CNN, utilizando el *script* de re-entrenamiento. Al modelo generado se le debe aplicar un segundo *script* para adaptar el modelo generado a su uso en la aplicación final. Finalmente, se debe copiar el modelo a la carpeta *assets* del proyecto Android de la aplicación, re-emplazando el modelo anterior.

4. Una vez hecho esto, se deben obtener tres imágenes donde se vea clara y exclusivamente el objeto, en el ámbito donde va a ser reconocido. A continuación, se deben copiar estas imágenes a la carpeta *drawable* del proyecto Android de la aplicación.

5. Para añadir la visualización aumentada, se deben obtener tres imágenes relacionadas con el objeto a añadir, y de la misma manera que antes, copiarlas a carpeta *drawable* del proyecto Android de la aplicación.

6. Con todos los pasos anteriores, solo se debe modificar el código de la aplicación en las secciones de carga de imágenes, tanto de visualización como de referencia, para cargar las nuevas imágenes, y asociarlas al nuevo objeto añadido. Como último paso, se debe añadir a la lista de nombres de objetos el nombre del nuevo objeto añadido.

Bibliografía

[1] David G Lowe. Object recognition from local scale-invariant features. In *IEEE Proc. of Int. Conf. on Computer Vision*, volume 2, pages 1150–1157, 1999.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[3] Jared Heinly, Enrique Dunn, and Jan-Michael Frahm. Comparative evaluation of binary features. In *Computer Vision–ECCV 2012*, pages 759–773. Springer, 2012.

[4] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[5] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[6] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*, volume 1. Cambridge: MIT press, 2016.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] TensorFLow. Main page of tensorflow `https://www.tensorflow.org`.

[10] TensorFLow. Mobile version of tensorflow `https://www.tensorflow.org/lite/tfmobile/android_build`.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[12] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[14] TensorFLow. Tensorboard: visualizing learning `https://www.tensorflow.org/guide/summaries_and_tensorboard`.

[15] EINA. Página web oficial del museo de informática, alojado en el edificio ada byron de la universidad de zaragoza. `http://mih.unizar.es/`.

[16] Google. Arcore `https://developers.google.com/ar/`.