UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Integración de redes neuronales en sistemas empotrados. Clasificación de imagen con RaspberryPi.

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Andreu Crespo Barberá

Tutor: José Enrique Simó Ten

Curso 2019-2020

Resum

Els sistemes encastats basats en plaques de grandària reduida i baix cost però de gran potència, incorporen una gran quantitat de dispositius que els fan especialment atractius per al desenvolupament d'aplicacions en entorns industrials.

En aquest Treball Fi de Grau es pretén desenvolupar un prototip de sistema encastat que permeta l'adquisició d'imatges generant un conjunt d'elles (*Data Set*) per a ser utilitzades en un procés d'entrenament basat en aprenentatge profund (*deep learning*) i generar un model de xarxa neuronal per a ser utilitzat per al reconeixement de noves peces.

El prototip a desenvolupar estarà format per dues aplicacions independents que s'executaran en computadors diferents connectats a través de la xarxa. Les aplicacions integren una capa de programari d'intermediació (*middleware*) que facilita la comunicació entre aquestes aplicacions. Una d'elles s'executarà en un sistema de propòsit general i, a través d'una interfície gràfica, l'operador interaccionarà amb l'altre computador per a l'adquisició i reconeixement de les imatges. L'altra aplicació s'executarà en un sistema encastat dedicat a l'adquisició d'imatges.

Aquest prototip s'utilitzarà en un entorn industrial de reconeixement de llepolies amb la finalitat d'etiquetar-les i controlar l'empaquetat.

Paraules clau: Sistema encastat, Informàtica industrial, interfície gràfica, visió per computador, aprenentatge, xarxes neuronals.

Resumen

Los sistemas empotrados basados en placas de pequeño tamaño y bajo coste pero de gran potencia, incorporan gran cantidad de dispositivos que los hacen especialmente atractivos para el desarrollo de aplicaciones en entornos industriales.

En este Trabajo Fin de Grado se pretende desarrollar un prototipo de sistema empotrado que permita la adquisición de imágenes generando un conjunto de ellas (*Data Set*) para ser utilizadas en un proceso de entrenamiento basado en aprendizaje profundo (*deep learning*) y generar un modelo de red neuronal para ser utilizado para el reconocimiento de nuevas piezas.

El prototipo a desarrollar estará formado por dos aplicaciones independientes que se ejecutarán en computadores diferentes conectados a través de la red. Las aplicaciones integran una capa de software de intermediación (*middleware*) que facilita la comunicación entre estas aplicaciones. Una de ellas se ejecutará en un sistema de propósito general y, a través de una interfaz gráfica, el operador interaccionará con el otro computador para la adquisición y reconocimiento de las imágenes. La otra aplicación se ejecutará en un sistema empotrado dedicado a la adquisición de imágenes.

Este prototipo se utilizará en un entorno industrial de reconocimiento de golosinas con el fin de etiquetarlas y controlar el empaquetado.

Palabras clave: Sistema empotrado, informática industrial, interfaz gráfica, visión por computador, aprendizaje, redes neuronales.

Abstract

Embedded systems based on small, low-cost but high-powered boards, incorporate a large number of devices that make them especially attractive for the development of applications in industrial environments.

In this TFG, the aim is to develop a prototype of an embedded system that allows the acquisition of images generating a set of them (*Data Set*) to be used in a training process based on deep learning (*deep learning*) and to generate a neural network model to be used for the recognition of new parts.

The prototype to be developed will be formed by two independent applications that will be executed in different computers connected through the network. The applications integrate a layer of intermediary software (*middleware*) that facilitates the communication between these applications. One of them will be executed in a general purpose system and, through a graphic interface, the operator will interact with the other computer for image acquisition and recognition. The other application will run on an embedded system dedicated to image acquisition.

This prototype will be used in an industrial candy recognition environment for the purpose of candy labeling and packaging control.

Key words: Embedded system, industrial informatics, graphic interface, computer vision, learning, neural networks.

Índice

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

La informática industrial ha tenido en estos últimos años un gran avance con la incorporación de la tecnologías hardware y software de forma masiva. Los sistemas empotrados basados en placas de pequeño tamaño, pero de gran potencia, incorporan gran cantidad de dispositivos que los hacen especialmente atractivos para el desarrollo de aplicaciones en entornos industriales. Además, el coste muy bajo de estos sistemas ha hecho que se hayan incorporado en todos los niveles de los procesos industriales.

En este Trabajo Fin de Grado se pretende desarrollar un prototipo de sistema empotrado con visión para el reconocimiento de patrones de objetos de productos comestibles para ser integrado en una línea de fabricación. El uso de redes neuronales en equipos de bajo coste ofrece unas grandes posibilidades de identificación y control en las líneas de producción. Este entorno es el marco del presente trabajo.

1.1 Motivación

Este trabajo Fin de Grado se enmarca en las actividades del Instituto de Automática e Informática Industrial (AI2) y su relación con las necesidades de las empresas. En sus contactos con las muchas empresas con las que se tienen relaciones, surgió la problemática de la identificación y etiquetado tanto de productos como de defectos y características superficiales en procesos de fabricación. De las problemáticas identificadas se escogió, por singularidad, una que requería identificar en una bolsa con productos comestibles ("chucherías") el número de elementos de cada tipo que se habían envasado. La necesidad de que las bolsas contengan el número determinado de cada tipo de producto es fundamental para el cliente.

Ante esta problemática, en el marco de AI2, se plantea la realización de TFG que aborde este problema analizando las distintas alternativas en cuanto a técnicas de reconocimiento de componentes y el desarrollo de un prototipo de sistema de visión empotrado que permita valorar la tecnología de reconocimiento y análisis.

Este TFG se ha realizado con una acción de prácticas en empresa en el AI2 en el periodo de enero a septiembre de 2019.

En este marco del TFG, el objetivo personal es poder poner en práctica los conocimientos adquiridos durante los estudios del Grado de Ingeniería Informática y, en especial, los conocimientos entre otros sobre sistemas distribuidos, desarrollo de aplicaciones, interfaces de usuario, técnicas de IA, etc.

En un TFM previo [Loz18] se abordó la problemática de la generación de imágenes de forma artificial y se realizó un estudio las técnicas de redes neuronales profundas para

su reconocimiento. Fundamentalmente se basa en la generación de imágenes de forma automática lo que simplifica enormemente el problema de la adquisición, tratamiento y reconocimiento de imágenes adquiridas mediante una cámara tal y como se realiza en el presente TFG.

1.2 Objetivo del trabajo

De acuerdo con la motivación anterior, el objetivo de este proyecto es el diseño y desarrollo de un sistema empotrado de visión que pueda integrarse en un entorno de fabricación y que permita identificar los distintos componentes de una tipología concreta en un producto envasado.

El objetivo del proyecto se puede desglosar en subobjetivos que aborden las distintas características del sistema a desarrollar. Estos subobjetivos son:

- Selección y desarrollo de un sistema empotrado que ofrezca las características necesarias para el desarrollo del sistema.
- Selección de técnicas de visión por computador para la identificación de los componentes.
- Diseño de la arquitectura hardware y software que permita la gestión del sistema de forma remota.
- Creación de un conjunto de piezas para entrenar y validar el modelo.
- Diseño y desarrollo de las aplicaciones que permitan la captura de imágenes, su uso para el entrenamiento de una RN y la explotación del resultado de entrenamiento en el sistema empotrado.
- Evaluación del sistema desarrollado

1.3 Estructura de la memoria

Este documento se ha estructurado de manera que permita una lectura ordenada. En el capítulo, se describe el estado de la técnica en cuanto a sistemas empotrados y técnicas de análisis.

En el capitulo 2 se plantea el estado de la técnica en aquellas tecnologías nucleares al TFG: sistemas empotrados, sistemas operativos, plataformas de sistemas empotrados comerciales y un análisis de las técnicas de inteligencia artificial y en especial de las redes neuronales.

El capítulo 3 se aborda el diseño del sistema mediante la definición de los casos de uso del sistema global. A partir de los casos de uso se han elaborado el conjunto de requisitos que guían el proceso de diseño del sistema.

El capítulo 4 presenta el diseño del sistema planteado. La visión global y la descomposición del sistema en componentes interconectados con interfaces claramente identificadas. En este capítulo se detallarán las técnicas de aprendizaje profundo que han sido utilizadas el desarrollo.

En el capítulo 5 se aborda la implementación del sistema. Se desarrollan dos soluciones basadas en redes neuronales que siendo básicamente muy iguales, representan visiones a evaluar. Por un lado el proceso de generación de un modelo de red neuronal

de aprendizaje profundo desarrollado en un lenguaje como R que permite sobre herramientas como Keras elaborar un modelo. Por otro lado, la utilización del lenguaje Python para prácticamente lo mismo, generación del modelo. Python, a diferencia de R, es un lenguaje mucho mas extendido y su uso industrial mucho mas aceptado. Se compararan los resultados obtenidos.

En el capítulo 6 se describe la experimentación realizada con un conjunto de imágenes adquiridas en el prototipo desarrollado generando un modelo de red neuronal y haciendo pruebas de reconocimiento de imágenes.

En el capítulo 7 se recogen las conclusiones del trabajo realizado. En el capítulo 8 se detallan algunas de los acciones que pueden ser objeto de trabajo futuro.

CAPÍTULO 2
Estado del arte

En este capítulo de detalla los principales aspectos relativos a las dos líneas del trabajo: los sistemas empotrados y las redes neuronales. Los sistemas empotrados como arquitectura hardware-software base para el desarrollo del prototipo se analizará y se compararán distintas placas. Por otro lado, las redes neuronales como técnica de reconocimiento de patrones que se integrará en la aplicación a desarrollar.

2.1 Sistemas empotrados

Un sistema empotrado o embebido es un sistema informático cuyo hardware y software están especialmente diseñados para resolver una problemática concreta en un entorno concreto de forma independiente o coordinada con otros subsistemas. El término empotrado o embebido obedece a que éste forma parte de un sistema mas grande formado por varios subsistemas y realiza una función determinada. Algunos ejemplos de estos sistemas se pueden encontrar en los automóviles, trenes, aviones, satélites, etc. Una característica común de los sistemas empotrados es la interacción con el entorno a través de sensores y actuadores.

La gran versatilidad y usabilidad de los sistemas empotrados, así como la diversidad de las interfaces disponibles, permite su aplicabilidad en una cualquier sector industrial aportando un gran valor añadido a los productos que lo incorporan. Es por ello, que el desarrollo de estos sistemas es un área estratégica para muchas empresas con el objetivo de aumentar su competitividad [OPT09].

2.1.1. Arquitectura del sistema empotrado

Un sistema empotrado consiste en uno o más unidades de cómputo (CPUs) conformando sistemas de un solo procesador (monocore) o con varios (multicore). Además el sistema tiene los componentes típicos de un computador como son la memoria, buses, dispositivos de entrada-salida, etc.

La figura 2.1 muestra la arquitectura básica de un sistema empotrado.

Aunque existen muchos sistemas empotrados basados en monocore, la tendencia es que estos sean multicore. El incremento de potencia de cómputo que se obtiene con varios procesadores permite que las aplicaciones que se ejecuten sean más potentes e incorporen técnicas que con los sistemas monocore requerirían tiempo de respuesta altos. La mejora sistemática de los procesadores permite obtener una elevada fiabilidad, mayor rendimiento, menor consumo energético, menor tamaño y menor coste.

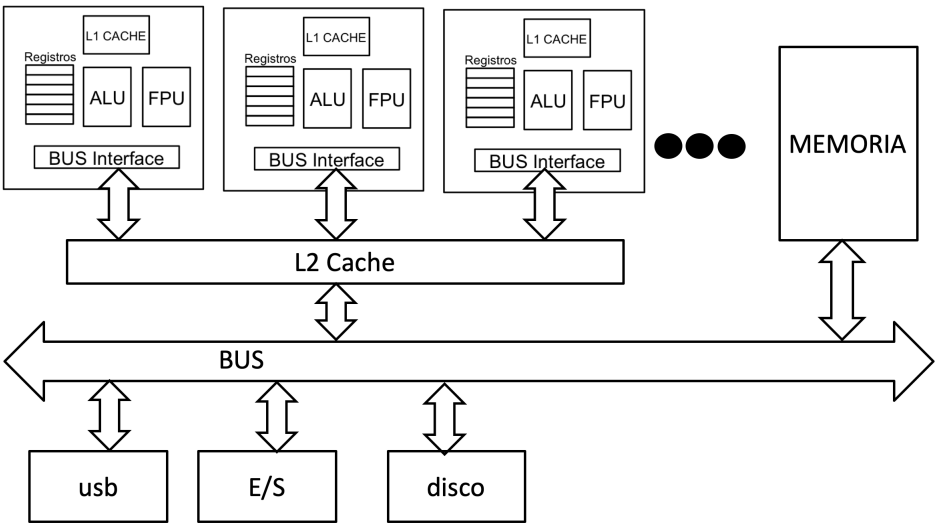


Figura 2.1: Sistema empotrado multicore

La arquitectura de un sistema empotrado está formada por los mismos elementos que conforman un sistema informático de propósito general con la adición de otros específicos para entrada y salida. A continuación se detallan estos componentes.

- CPU: la unidad central de proceso es el elemento encargado de ejecutar las instrucciones de las tareas que conforman las aplicaciones. Actualmente, como se ha comentado anteriormente, los sistemas están integrados por varios CPUs definiendo sistemas multi-núcleo o multicore. Cada unidad de procesamiento (núcleo) aporta una unidad de cómputo que, con las otros CPUs, permite la ejecución paralela. La mayoría de sistemas empotrados comerciales son multi-núcleo compartiendo la memoria. Cada CPU tiene

- Memoria: La memoria permite almacenar los programas en ejecución o procesos junto con los datos. Es uno de los componentes críticos ya que impacta en dos aspectos: el tamaño y el tiempo. En cuanto al tamaño, permite tener más o menos programas en ejecución y más o menos regiones de datos de éstos. El tamaño de los datos puede ser muchas veces crítico en aplicaciones donde se necesitan grandes volúmenes de información. Por ejemplo, las aplicaciones de tratamiento de imágenes puede tener un fuerte impacto. Por otro lado, los tiempos de acceso y de escritura en la memoria influyen decisivamente en los tiempos de ejecución de los programas.

- Cache: Memoria de acceso rápido que permite almacenar código y datos accedidos recientemente por las CPUs. Tiene un fuerte impacto en el tiempo de ejecución de los programas. Cada CPU dispone de una cache local (L1) en donde se almacenan el código y datos del proceso en ejecución esa CPU. A otro nivel, la cache compartida (L2) permite almacenar también código y datos de los procesos en ejecución en las CPUs. Tamaños de cache (L1 y L2) y políticas de reemplazo y ubicación permiten incrementar la velocidad de cómputo de los procesos.

- Almacenamiento externo: El almacenamiento del sistema de ficheros con los programas, utilidades, ficheros de arranque y ficheros en general, se dispone en unidades de disco externos. En los sistemas empotrados, lo más común, es que estos dispositivos sean discos de estado sólido (SSD) con una capacidad limitada y removibles. En los sistemas empotrados, el disco contiene los ficheros de arranque del

sistema, ficheros de configuración de las aplicaciones, ejecutables de las aplicaciones y ficheros de datos.

- Dispositivos: Existen un gran número de dispositivos que se utilizan en los sistemas empotrados orientados a comunicaciones, entrada salida digital y analógica, visualizadores de 7 segmentos y LCD, teclados específicos, acelerómetros, sensores de distintos tipos, etc. En la mayoría de casos, estos dispositivos conectables a los sistemas empotrados disponen de una interfaz estándar (PMOD, GPIO, y otras) con la que se conecta a través de un conector estándar.

2.2 Sistemas operativos

Uno de los elementos clave en los sistemas empotrados es el sistema operativo. Los sistemas operativos clasificados de acuerdo al tipo de aplicaciones a soportar (por ejemplo de propósito general, distribuidos, tiempo real, sistemas empotrados, sistemas paralelos, etc.). Sin embargo, en los últimos años, las diferencias entre éstos se han difuminado, existiendo sistemas operativos que pueden cubrir diferentes tipos de aplicaciones. En el sector de la automatización, generalmente, los sistemas operativos más usados entran en los grupos de sistemas de tiempo real, empotrados y de propósito general [CA06]

Los sistemas operativos de propósito general tienden a gestionar los recursos del sistema (CPU, memoria, dispositivos) de una forma equitativa entre los usuarios y/o las aplicaciones. Suelen disponer de una interfaz de usuario gráfica para una administración y programación más sencilla e intuitiva. Ofrecen aplicaciones de carácter general. Ejemplos de este tipo de sistemas operativos son: Linux, Windows, MacOS, FreeBSD, OpenBSD, etc.

Los sistemas operativos de tiempo real son un tipo especial de sistema operativo en el que no se persigue un reparto equitativo de los recursos del sistema sino apropiado a los requisitos de las aplicaciones. Garantizan el cumplimiento de las restricciones temporales (periodicidad, tiempo de respuesta, robustez, etc.). Los componentes de un sistema operativo de tiempo real ofrecen un comportamiento predecible y determinado. Asimismo, las políticas de planificación de las tareas se realiza mediante prioridades estáticas o dinámicas.

Los sistemas operativos empotrados son sistemas operativos adaptados a las necesidades específicas de un hardware en el que va a operar junto con otros subsistemas. Normalmente hay unos factores de simplificación que los diferencian de los sistemas operativos tradicionales:

- Un único usuario: No es necesario disponer de varios usuarios, es un desarrollo para funcionar con unas características determinadas.
- No suelen llevar los dispositivos clásicos: ratón, pantalla, teclado, etc. Éstos son sustituidos por dispositivos específicos: visualizadores, teclados especiales, pulsadores, etc.
- Normalmente, se cargan desde memoria flash o disco SSD. Cuando lo requieren, se suelen usar memoria del tipo flash como sistema de ficheros.
- En un sistema empotrado no se realizan operaciones de desarrollo del programa. Esto evita disponer de editores, compiladores, etc. El desarrollo se realiza en otra máquina distinta que puede llevar un distinto procesador (desarrollo cruzado). En este caso, cuando se completa el proceso de desarrollo, el resultado, que incluye el

sistema operativo y las aplicaciones, se carga en el sistema empotrado. Modificaciones de la aplicación suelen requerir el reemplazo de todo el sistema.

- El número de aplicaciones o tareas es limitado y conocido. El sistema operativo para empotrados debe ser compilado con los parámetros de configuración ajustados a cada aplicación. Esto permite ajustar el tamaño del sistema operativo a unas necesidades concretas. Este proceso de configuración incluye la selección de los dispositivos disponibles, cantidad de memoria, número de tareas, tipo de planificador, etc.

La tremenda evolución del hardware ha permitido que sistemas operativos de propósito general se adapten a los sistemas empotrados. Existe un gran número de distribuciones del sistema operativo Linux adaptadas para sistemas empotrados con distintos procesadores. Las principales ventajas del uso de un sistema empotrado basado en Linux se pueden resumir en los siguientes puntos:

- Disponibilidad de compiladores para varios lenguajes.
- Existencias de gran número de herramientas tanto para el desarrollo de las aplicaciones como para la construcción y desarrollo del sistema empotrado.
- Herramientas para la depuración de las aplicaciones.
- Gran diversidad de manejadores de dispositivos y protocolos de comunicación.
- Disponibilidad de parches para el para sistemas de tiempo real
- Una gran comunidad de desarrolladores y usuarios.

Algunas de las distribuciones más usadas son:

- Debian: también conocida como Debian GNU/Linux, es una distribución de Linux compuesta por software libre y de código abierto. La distribución estable es muy popular y es usada por otras distribuciones como base.
- Ubuntu: basada en Debian, es una distribución muy popular tanto para servidores, escritorio o empotrados.
- Android es una distribución de sistema operativo para móviles basada en una versión modificada de Linux.

Además de estas, existen otras muchas como RedHat, CentOS, Fedora, openSUSE, etc. También plataformas de ejecución ofrecen sus propias distribuciones como es el caso de la Raspbian [Pi17].

2.2.1. Software intermedio

Las comunicaciones en los sistemas industriales se han desarrollado principalmente en torno a tecnologías específicas de cada sector. Sin embargo, la evolución tecnológica se caracteriza por el alto grado de interconexión entre los elementos que forman parte del sistema industrial.

Este nuevo volumen de información requiere adoptar nuevas tecnologías de comunicación que sean abiertas, interoperables, eficientes y que sean capaces de proporcionar

la calidad de servicio requerida. Tecnologías de comunicaciones basadas en un software intermedio o middleware que están llamadas a desempeñar un papel relevante en la interconexión e integración de la próxima generación de sistemas industriales.

Por middleware se entiende la capa intermedia del software entre el sistema operativo y las aplicaciones que ofrece servicios de comunicación y distribución de la información entre las distintas aplicaciones. El middleware puede estar disponible en distintos sistemas empotrados que conectados a través de la red permite que las aplicaciones que se ejecutan en los distintos nodos de un sistema distribuido se comuniquen e intercambien información de una manera totalmente transparente. Un middleware proporciona servicios a las aplicaciones para que se identifiquen, autentifiquen y accedan a los datos disponibles en el middleware. Esta capa permite a las aplicaciones ubicarse en distintos nodos e intercomunicarse a través de interfaces normalizadas.

De los diversos tipos de middleware desarrollados solo la aproximación orientada a objetos y a mensajes ofrecen una funcionalidad completa para su uso en entornos distribuidos en tiempo real.

El middleware procedural o RPC (Remote Procedure Call) proporciona la capacidad de invocar una función o método para ejecutar en otro espacio de direcciones o nodo. El middleware orientado a mensajes (MOM) se usa para comunicar la información y los datos mediante mensajes. La interacción mediante publicación/suscripción (DCPS) para la gestión de mensajes ofrece un mecanismo muy útil y eficiente para distribuir información. En este mecanismo, los editores publican los mensajes y los entregan a los solicitantes que se han suscrito para recibir esos mensajes. La figura 2.2 ofrece un esquema de la relación de las aplicaciones y el middleware .

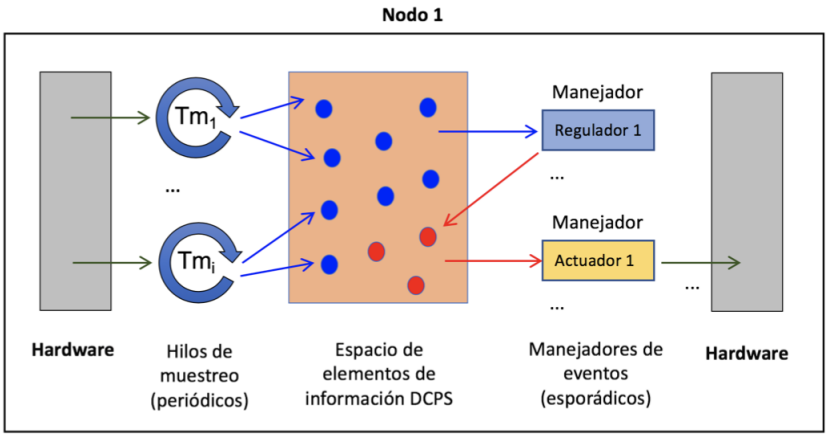


Figura 2.2: Modelo de aplicación Middleware. Fuente: [Sim18]

Existen algunos middleware como MQTT [BG14] que ofrece un protocolo de comunicaciones basado en el paradigma de publicación/suscripción y que se caracteriza principalmente por ser ligero y abierto, DDS [Gro15] como estándar de la OMG para interacción entre entidades distribuidas basado igualmente en un paradigma de publicación/-suscripción y OPC [Fou15] es un estándar industrial que busca facilitar la integración de dispositivos y aplicaciones en el control de la producción. Basado en DDS, el multipeer [Sim18] ofrece los servicios de DDS con una alta capacidad de respuesta en tiempo real.

2.3 Plataformas comerciales

Existe un número significativo de placas comerciales de costo reducido para el desarrollo de sistemas empotrados. Estas placas ofrecen gran diversidad de dispositivos para

conectarse con elementos externos y gran capacidad de cómputo. A continuación de detallan y comparan las tres placas más comunes y de precio reducido para el desarrollo de este tipo de sistemas.

2.3.1. Beagle Bone

Beagle Bone es el nombre comercial de un conjunto de placas con distintas características de con alto rendimiento y coste reducido. Existen varias placas con distintas características nombradas como Blue, Black, AI, etc. Por ejemplo, la mas reciente es la AI orientada a altas prestaciones para aplicaciones con técnicas de inteligencia artificial que está basada en el procesador de Texas Instruments Sitara AM5729 configurando un sistema heterogéneo con un subsistema formado por 2 cores ARM A15, 2 cores de ARM M4, y 2 cores PowerVr de procesamiento gráfico.

Además ofrece una memoria RAM de 1GB e interfaz eMMC flash y otros dispositivos como Gigabit Ethernet, 2.4/5GHz WiFi, Bluetooth y microHDMI. La figura 2.3 ofrece un diagrama de bloques de la placa.

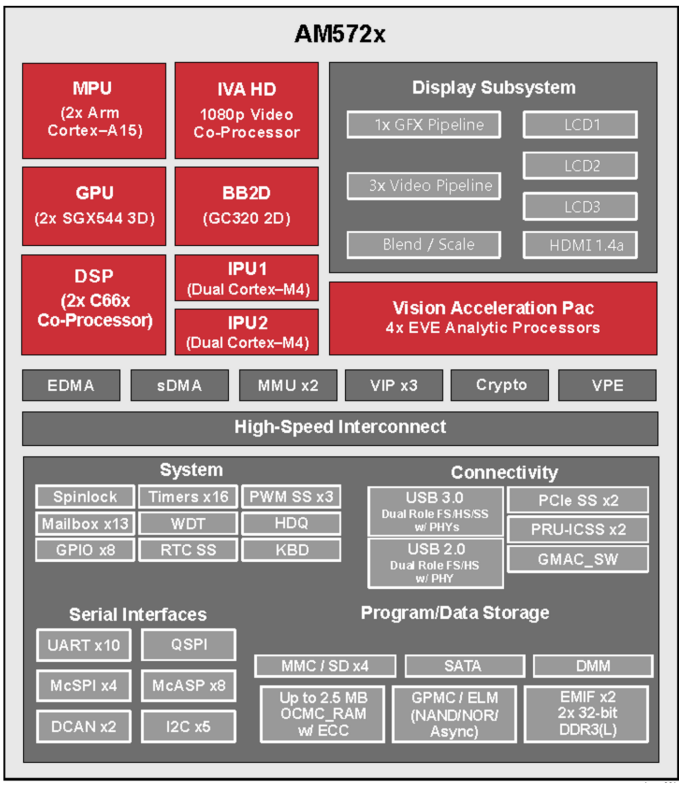


Figura 2.3: Diagrama de bloques del AM5729. Fuente: Texas Instruments

Existe una distribución de Linux basado en Debian para todas las distintas placas Beagle Bone.

El coste de la placa Beagle Bone AI es de unos 120 euros.

2.3.2. Raspberry Pi

Raspberry Pi es la marca de un conjunto de desarrollos de placas para empotrados de bajo costo y grandes prestaciones. Desarrollado inicialmente para fomentar la docencia de la informática en centros educativos del Reino Unido, se ha consolidado como un producto de amplia aplicación en sistemas empotrados industriales. La distintas evoluciones

del desarrollo inicial han incluido nuevas características y dispositivos. La versión actual de la placa Raspberry Pi 4 esta basada en el chip Broadcom BCM2711 que está formado por cuatro núcleos del ARM Cortex-A72 (ARM v8) de 64-bit en SoC a 1.5GHz, dispone de distintas versiones de memoria RAM (1GB, 2GB or 4GB) , 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE, Gigabit Ethernet, 2 USB 3.0 ports; 2 USB 2.0 ports. Además, ofrece 40 pin GPIO compatible con versiones previas y otros dispositivos como 2 x micro-HDMI ports , 2- líneas de display port, OpenGL ES 3.0 graphics, Micro-SD, etc.

La figura 2.4 muestra el diagrama de bloques de la placa.

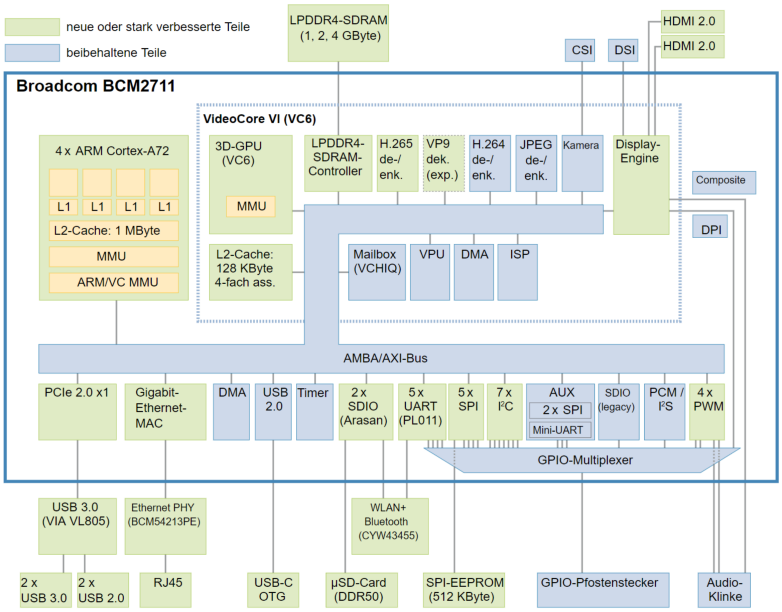


Figura 2.4: Diagrama de bloques del chip Broadcom BCM2711. Fuente: Raspberry Pi web

La web de Raspberry Pi ofrece a la comunidad de desarrolladores una distribución de Linux basada en Debian para su descarga llamada Raspbian. Asimismo, se ofrecen otras distribuciones como Ubuntu, Windows 10 IoT Core o RISC OS.

Existen un gran número de dispositivos conectables a las Raspberry Pi a través de los conectores de GPIO como cámaras, pantallas, etc.

El precio de la placa Raspberry Pi 4 es de 55 euros.

Además de esto la Raspberry dispone de un interfaz hardware para la captura de imágenes a alta velocidad y cámaras asequibles en precio. El rango de precios de la cámara está entre 10 y 30 euros.

2.3.3. Arduino

Arduino es una familia de microcontroladores que han tenido un fuerte impacto en el desarrollo de pequeños sistemas empotrados. Uno de los mas potentes es el Arduino Zero, un desarrollo conjunto entre Arduino y Atmel, que es una extensión de 32 bits de la plataforma establecida por Arduino UNO. El objetivo es potenclas aplicaciones para dispositivos inteligentes de IO, IoT, automatización de alta tecnología, robótica y otros proyectos innovadores.

La placa está basada en el microcontrolador MCU SAMD21 de Atmel, que contiene un núcleo ARM Cortex M0+ de 32 bits. Dispone de distintos dispositivos para las comunicaciones Ethernet, puertos USB y convertidores analógico-digitales y señales digitales.

La figura 2.5 ofrece un diagrama de bloques de la placa.

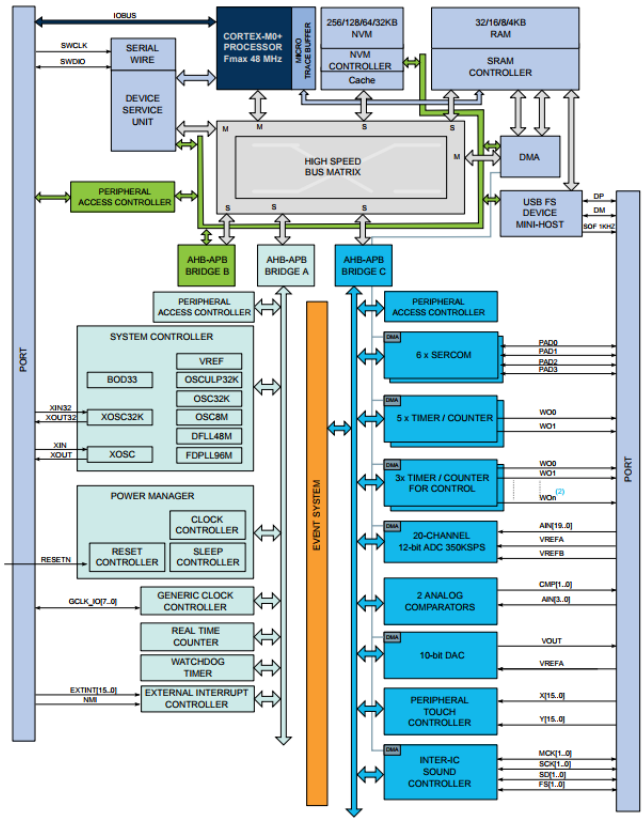


Figura 2.5: Diagrama de bloques del MCU SAMD21. Fuente: microchipdeveloper

Se ofrece una interfaz de programación (IDE) para el desarrollo de aplicaciones en distintos sistemas operativos.

El precio de la placa Arduino Zero es de 25 euros.

2.4 Inteligencia Artificial en la industria

La industria está sufriendo en estos tiempos una fuerte revolución impulsada por la nuevas tecnologías informáticas. Concretamente, la internet de las cosas (*Internet of Things IoT*) es fundamental en la transformación digital de la industria. Tanto a nivel europeo como mundial, se ha extendido el concepto de Industria 4.0 para denominar la que se considera cuarta revolución industrial en el que los sistemas informáticos y, concretamente, los sistemas empotrados juegan un papel fundamental.

Estas tecnologías, identificadas como habilitantes, definen un marco tecnológico que la industria 4.0 ha asumido. Como tecnologías habilitantes se consideran aquellas que permiten integrar, interconectar, procesar y analizar la información masiva de información generada en la industria. Algunas de ellas son la IoT y la integración de sensores que permiten adquirir la información de distintos niveles del proceso industrial, la monitorización y control en tiempo real de las líneas de producción, la robótica como elemento capaz de realizar acciones sobre el proceso, la integración de la información horizontal mediante *middleware* y vertical, el procesamiento en la nube y la aplicación de técnicas de Big Data a la información, la simulación de elementos del proceso, la realidad aumentada y otros tecnologías mas específicas.

En este ámbito y también a nivel general, las técnicas de inteligencia artificial se identifican como fundamentales para la transformación de los productos, procesos y modelos de negocio en todos los sectores económicos. Durante las últimas décadas, la IA ha sido el centro de una intensa investigación que ha logrado alcanzar unos niveles muy altos en cuanto a resultados en el procesamiento masivo de información, el reconocimiento de la voz, la traducción a partir de la voz, el reconocimiento de objetos y patrones a partir de la imagen, la predicción de nuevas situaciones, etc.

Una de las técnicas de IA es el aprendizaje automático (*machine learning*) que persigue que el computador aprenda por sí mismo a a realizar una determinada actividad. El aprendizaje automático permite, a partir de unos datos y los resultados a los datos, generar un conjunto de reglas a aplicar a nuevos datos para producir nuevas respuestas. El aprendizaje automático se realiza mediante un proceso de entrenamiento mediante ejemplos relevantes y se genera una representación determinada capaz de generar las reglas.

El aprendizaje profundo (*deep learning*) es un subcampo del aprendizaje automático entendido como una nueva forma de entender las representaciones del aprendizaje. Se basa en el modelado mediante varias capas sucesivas de representaciones que razonan de una forma más detallada (profunda) sobre un proceso. El número de capas determina la profundidad del modelo.

Una de las técnicas usadas para el aprendizaje profundo es la basada en redes neuronales utilizada para la identificación y reconocimiento de patrones. Las redes neuronales son actualmente una herramienta que se puede aplicar a una variedad de problemas. La clasificación y la regresión son algunas de los problemas más generales pero también se utilizan en el control, el modelado, la predicción y el pronóstico.

Como resultado de la investigación en redes neuronales, se dispone de un gran conjunto de herramientas de software utilizadas para entrenar y trabajar en este tipo de redes. En los siguientes apartados se introduce el concepto de red neuronal y se detallan las herramientas disponibles.

2.4.1. Redes neuronales

Las redes neuronales (RN) son un modelo computacional que debe su nombre a la idea de las redes neuronales de los organismos vivos: un grupo de neuronas conectadas entre sí y que sin que haya una tarea específica para cada una son capaces de trabajar en grupo. Las neuronas están conectadas mediante enlaces. Con la experiencia, las neuronas van reforzando las conexiones para "aprender" algo. La información se somete a diversas operaciones conforme atraviesa la red neuronal produciendo un valor de salida.

La característica principal de este sistema es que aprende y se forma a sí mismo en vez de ser programado de forma explícita. Para realizar el aprendizaje autómatico se utiliza un método llamado propagación hacia atrás (*backtracking*). Consiste en ir actualizando los valores de los pesos de las neuronas para reducir el valor de la función de pérdida.

Una de las aplicaciones típicas de las redes neuronales es el reconocimiento de objetos en imágenes (visión por computador) o el reconocimiento automático de la voz.

A continuación se detallan algunos de los tipos de red neuronal más utilizados. Para una información más detallada se recomienda el libro Redes Neuronales y *Deep Learning* [Ber18].

- Perceptron (1958): el cíentifico Frank Rosenblatt crea el perceptron, la unidad donde nacerían y se potenciarían las redes neuronales artificiales. Un perceptron toma

varias entradas binarias (x) y produce una salida binaria (y). Para poder calcular la salida, se introduce el concepto de "pesos"(w), un número real que expresa la importancia de la respectiva entrada con la salida. La salida será 1 o 0 si la suma de la multiplicación de pesos por entradas es mayor o menor a un determinado umbral.

$$y = \sum_i x_i y_i = x_1 w_1 + x_2 w_2 \tag{2.1}$$

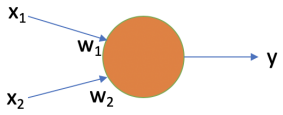


Figura 2.6: Esquema de un Perceptron

- Multilayer perceptron (1965): es una ampliación de perceptron de una única neurona a más de una. También se incluye el concepto de capas de entrada, oculta y salida pero con entradas valores de entrada y salida binarios.

$$y = \sum_i x_i y_i = x_1 w_1 + x_2 w_2 + ... + x_n y_n \tag{2.2}$$

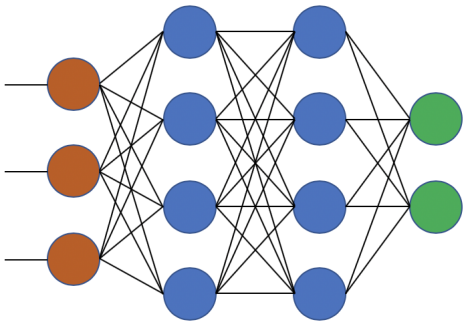


Figura 2.7: Multiplayer perceptron

- Neuronas sigmoides (1981): son parecidas al perceptron pero permite que las entradas tener valores reales. En este caso la la salida en vez de ser 0 o 1:

$$d(w * (x + b)) \tag{2.3}$$

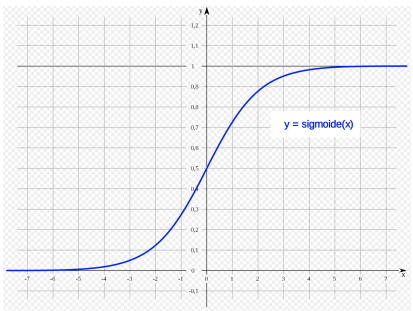


Figura 2.8: Función sigmoide

- Redes *feedforward* (1983): redes donde la salida las salidas de una capa son la entrada de la siguiente capa.

- *Backpropagation* (1986): al calcular el error obtenido en la salida e ir propagando hacia detrás se van haciendo pequeños ajustes en cada iteración para conseguir que la red aprenda, con esto la red podría, por ejemplo, clasificar las entradas correctamente.

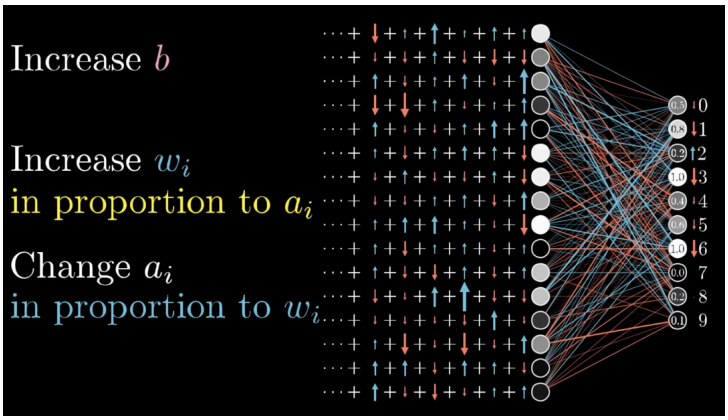


Figura 2.9: Backpropagation. Fuente: gfycat.com/equatorialspicyegg-mathematics-3b1b

- Redes convolucionales (1989): esta arquitectura es muy útil en el procesado de imágenes. Consta de varias capas que implementan la extracción de características para su clasificación. La imagen se parte en campos que alimentan una capa convolucional que extrae las características (*features*) de la imagen de entrada. Más tarde se realiza la agrupación (*pooling*) que reduce la dimensión de las características extraídas de las imágenes, manteniendo la información importante. Luego se repite el proceso para alimentar una red *feedforward* multicapa. La salida final de la red son nodos capaces de clasificar el resultado.

  Esta arquitectura usando capas profundas y la clasificación de salida abrieron un mundo nuevo de posibilidades en las redes neuronales.

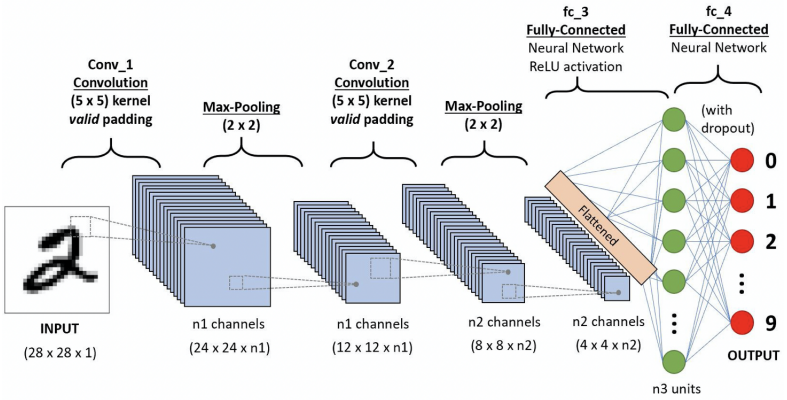


Figura 2.10: Redes convolucionales. Fuente: researchgate.net.

- Deep Belief Networks (DBN) (2006): antes del 2006 los modelos profundos (decenas o cientos de capas) se consideraban difíciles de entrenar y el uso de redes neuronales artificiales quedó estancado. Las DBN, demostraron que es una mala idea utilizar pesos aleatorios al inicializar las redes: por ejemplo, al utilizar Backpropagation con descenso por gradiente podía pasar que el aprendizaje cayera en mínimos locales, sin lograr optimizar los pesos. En este caso lo mejor sería utilizar una asignación de pesos inteligente mediante un pre-entrenamiento de las capas de la red. Esta idea se basa la utilización de Restricted Boltzmann Machines y Autoencoders para pre-entrenar la red de manera no supervisada. Luego de pre-entrenar y

asignar esos pesos iniciales, se debe entrenar la red de forma habitual, supervisada (por ejemplo con backpropagation). Se cree que ese pre-entrenamiento es una de las causas de la gran mejora en las redes neuronales pues para asignar los valores se evalúa capa a capa, de a una, y no «sufre» de cierto sesgo que causa el backpropagation, al entrenar a todas las capas en simultáneo.

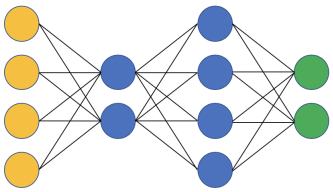


Figura 2.11: Redes Deep Belief Networks

En la actualidad, existen nuevos estudios de las neuronas humanas biológicas en las que se está redescubriendo su funcionamiento y se está produciendo una nueva revolución, pues parece que es totalmente distinto a los procedimientos actuales. Esto puede ser el principio de una nueva etapa totalmente nueva y seguramente mejor del Aprendizaje Profundo, el Machine Learning y la Inteligencia Artificial.

Existen diferentes técnicas y otras evoluciones tecnológicas importantes que no son mencionadas. Se comentan las más significativas junto con las relacionadas con las técnicas utilizadas para este trabajo.

2.4.2. Herramientas

La programación de las redes neuronales requiere una infraestructura importante para alcanzar un nivel de calidad y eficiencia requerido para las aplicaciones industriales. Esto puede implicar el abordar distintas módulos de programación a nivel de lenguaje de bajo nivel con el fin de acelerar las operaciones básicas o el uso de hardware dedicado para ello. Como resultado del trabajo de distintos grupos de investigación y desarrollo se dispone de herramientas de libre disposición para su integración y uso en aplicaciones que lo requieran. La utilización de estas herramientas se puede realizar desde cualquier lenguaje de programación aunque muchas de ellas se han integrado en entornos de desarrollo basados en lenguajes interpretados como Python o R de alto poder expresivo y que puede ser ejecutados en cualquier sistema operativo.

Tensorflow

Tensorflow es una librería de software de código abierto diseñada para realizar el cálculo de vectores (tensores) numéricos de una forma muy eficiente. Tensorflow has sido desarrollado por el equipo de Google Brain que pone a disposición de los desarrolladores la librería para el desarrollo de aplicaciones sobre distintos procesadores tanto de propósito general (CPU), como gráficos (GPU) o específico (TPU). TPU (Tensor Processing Unit) es una unidad de hardware creada por Google para procesar tensores directamente. Es un hardware próximo a lo que sería una FPU (Floating Point Unit) y que consigue una eficiencia muy alta en el procesamiento de vectores.

La interfaz de Tensorflow proporciona las funciones necesarias para el proceso de aprendizaje automático y profundo. Se ha desarrollado en C/C++ y ofrece interfaces (APIs) con lenguajes como Python, JavaScipt, Java, R y otros. En este momento, solo la integración es estable.

Una ventaja adicional de Tensorflow es su compatibilidad con plataformas de computación en la nube como la plataforma de nube de Google y Amazon Web Services. Esto lo hace muy interesante para aplicaciones o modelos que se desplieguen en servidores en la nube.

Theano

Theano es una de las primeras librerías desarrolladas para el aprendizaje profundo. Es una librería de Python y un compilador optimizado para el cálculo de tensores. Theano utiliza la sintaxis numpy (librería numérica de Pyhton) que se compila para que se ejecute de forma eficiente tanto en la CPU como en la GPU. Es un software de código abierto desarrollado la Universidad de Montreal. Es una biblioteca fundamental para el aprendizaje profundo que se puede utilizar directamente para crear modelos de aprendizaje profundo. Theano puede usar el hardware disponible en la máquina (CPU, GPU) con el fin de optimizar el código y acelerar la ejecución.

Aunque Theano tiene una sintaxis compleja, su uso en entornos de mayor nivel puede facilitar su utilización.

Keras

Keras es una biblioteca de aprendizaje profundo escrita en Python. Presenta una interfaz de fácil uso que oculta las complejidades internas. Keras puede usarse junto con Tensorflow o Theano. Es uno de los entornos más utilizado tanto en los desarrollos de investigación como los de producción.

Keras dispone de bloques de redes neuronales mas usados como capas, objetivos, funciones de activación, optimizadores, y añade herramientas para el manejo de imágenes y texto. Keras también soporta capas convolucionales y redes neuronales recurrentes.

Para los desarrolladores, que sólo quieren agregar funcionalidades de aprendizaje profundo a sus aplicaciones, Keras es una buena opción ya que le permite desarrollar aplicaciones a un ritmo mucho más rápido.

Torch/Pytorch

Torch es una librería para el cálculo de tensores multidimensionales. Torch ha sido desarrollado por Facebook y usado por este, Twitter y Google. Usa CUDA sobre la GPU.

Pytorch es un entorno de programación para aprendizaje profundo en el lenguaje Python que utiliza la librería Torch. y ha sido muy bien recibida por la comunidad de aprendizaje profundo. Pytorch permite utilizar modelos de aprendizaje profundo desde Python de gran eficiencia.

CUDA

A diferencia de las herramientas anteriores, CUDA (Compute Unified Device Architecture) no es un entorno para modelado de redes neuronales pero puede ser utilizado por estas herramientas para acelerar las operaciones que se requieren. CUDA es una plataforma de ejecución para la computación paralela integrado por una libreria y un hardware específico integrado por una GPU. Fue creado por nVidia para acelerar los gráficos en el computador y extendido como libreria para el uso de distintas aplicaciones. Se puede usar desde lenguajes como C y C++ o con adpatadores en Python, Java R, y otros.

openCV

OpenCV es una librería de funciones para el procesamiento de imágenes. Está disponible para distintos sistemas operativos (Linux, Windows, MacOS, etc.) y está licenciada bajo licencia BSD. Desarrollada inicialmente por Intel para la evaluación de aplicaciones de alto uso de CPU, se ha convertido en una de las librerías mas usada para el proceso de imágenes por su versatilidad y eficiencia.

La librería dispone de un conjunto numeroso de funciones que cubren amplias necesidades de tratamiento de imágenes como el reconocimiento de objetos, visión en tiempo real, seguimiento de objetos, etc. Se utiliza en campos de investigación e industriales diversos: robótica, guiado automático, seguimiento de objetos, etc.

Desarrollada inicialmente en C y portada a C++, se puede utilizar desde distintos lenguajes como C, C++, Java y Python.

Python

Python es un lenguaje de programación interpretado de gran poder expresivo y que ofrece la posibilidad de utilizar programación orientada a objetos, imperativa y funcional. Al ser interpretado, se puede ejecutar en cualquier plataforma. El lenguaje dispone de listas, diccionarios, conjuntos, etc., lo que permite programar de forma rápida y eficiente. Dispone de un gran número de módulos que permiten desarrollar aplicaciones para ámbitos muy diversos. Uno de estos módulos es NumPy que es una librería para el cálculo con matricial y vectorial. Otros módulos, como Keras o Pythorch, están orientados al aprendizaje profundo.

R

R es un lenguaje de programación interpretado y el entorno asociado. Es un lenguaje de enfoque orientado a la estadística que tiene gran poder expresivo. Al igual que Python, ofrece las estructuras de listas, diccionarios, vectores y matrices. R ofrece un conjunto de herramientas estadísticas (modelos lineales y no lineales, análisis de series temporales, tests estadísticos, algoritmos de clasificación y agrupamiento, etc.) y herramientas gráficas. La sintaxis es muy similar a otros lenguajes como Matlab. Al igual que Python, existen un gran número de módulos para las aplicaciones más diversas.

Otras herramientas

Existen otras herramientas como Caffe *Convolutional Architecture for Fast Feature Embedding*, entorno de aprendizaje profundo de código abierto desarrollado en la Universidad de Berkeley, MXnet, entorno dearrollado por Apache, Microsoft Cognitive toolkit, y otros menos utilizados.

2.5 Conclusiones

En este capítulo se ha descrito el estado de la técnica de los temas que conforman este trabajo fin de grado. Por un lado el sistema empotrado que representa la plataforma hardware y software sobre la que se implementan las aplicaciones que utilizan las técnicas de inteligencia artificial basadas en redes neuronales para la clasificación y reconocimiento de piezas en un entorno industrial.

En cuanto a los sistemas empotrados, se ha descrito la evolución y estado actual identificando el tipo de hardware, los sistemas operativos y las plataformas comerciales mas apropiadas. Desde este punto de vista se ha optado por la selección de la placa Raspberry Pi 4 debido a las siguientes razones:

- La placa ofrece un potente sistema de cómputo con 4 núcleos de ARM-A72 de 64 bits en un SoC.
- El sistema de desarrollo basado en el sistema operativo Raspian (basado en la distribución de Debian) ofrece unas prestaciones excelentes.
- Dispone de un gran número de dispositivos conectables y soportados por manejadores de dispositivos incluidos en la distribución.
- El bajo coste de la placa, aunque similar a las otras opciones, junto con los componentes adicionales y la disponibilidad de cajas para su integración industrial es otro elemento decisivo.

En conclusión a la selección de la plataforma, el RaspBerry PI 4 ofrece la mejor relación prestaciones y herramientas respecto al coste de todas las opciones consideradas.

Desde el punto de los lenguajes y herramientas, se opta por:

- El modelado de la red neuronal con aprendizaje profundo se utilizará la librería Keras por ser la más eficiente y es fácilmente integrable en varios lenguajes de programación.
- La capa de aplicación para el modelado y generación de modelos, se usarán los lenguajes R y Python por su alto poder expresivo. Se usarán ambos ya que se quieren comparar las características y prestaciones de los entornos con ambos lenguajes.
- Se utilizará el lenguaje C++ para el desarrollo de la aplicación en el sistema empotrado para la captura de imágenes y reconocimiento de objetos.
- Para el middleware se utilizará software del AI2 desarrollado en C++.
- Para la transformación de modelos a javaScript se utiliza Frugally-deep. Esta herramienta es una pequeña biblioteca de sólo cabecera escrita en C++.

CAPÍTULO 3
Análisis del problema

Una vez finalizado el estudio sobre el estado del arte de tanto las plataformas hardware como el software (sistemas operativos, middleware y lenguajes y herramientas de programación, se aborda en este capítulo las necesidades de la aplicación a desarrollar en el marco de este TFG. Para ello, se elaborarán los requisitos de base que se utilizarán para el diseño y desarrollo del sistema con las distintas aplicaciones.

Los requisitos se abordan desde la perspectiva de un cliente que va a utilizar el sistema a desarrollar. Para ello, se plantea inicialmente los casos de uso del sistema lo que permite dirigir de forma sistemática el proceso de desarrollo de los requisitos. Inicialmente, se ofrece los principales elementos que configuran el sistema completo.

3.1 Visión del sistema global

La visión general de sistema global se esquematiza en la figura 3.2. En ella se identifica el nodo formado por el sistema de gestión (SG) en el que se interaccionará con el usuario y el sistema de adquisición y reconocimiento (SAR) de imágenes con el que se interactuará con los dispositivos físicos para la adquisición y reconocimiento de piezas. Como se puede ver en la figura, pueden existir distintos subsistemas SAR interactuando con el SG.

Las figuras 3.2 y 3.1 ofrecen un esquema global del sistema completo.

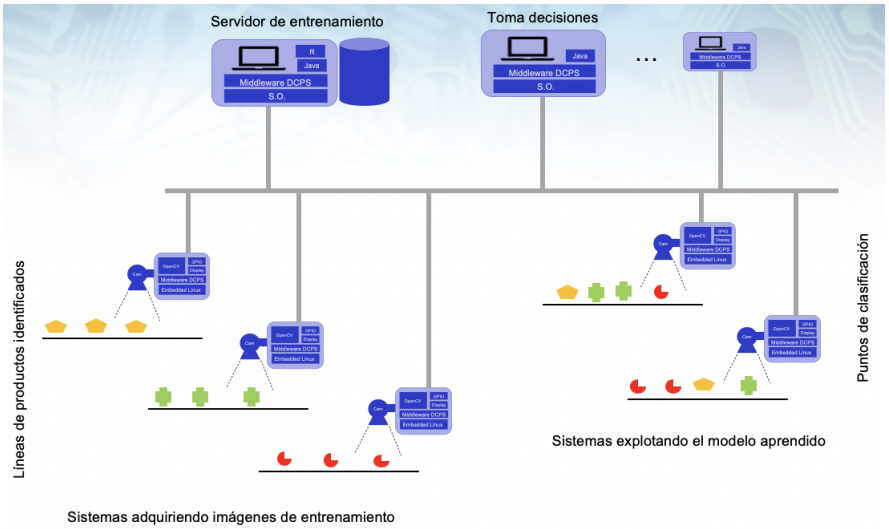


Figura 3.1: Esquema global del sistema

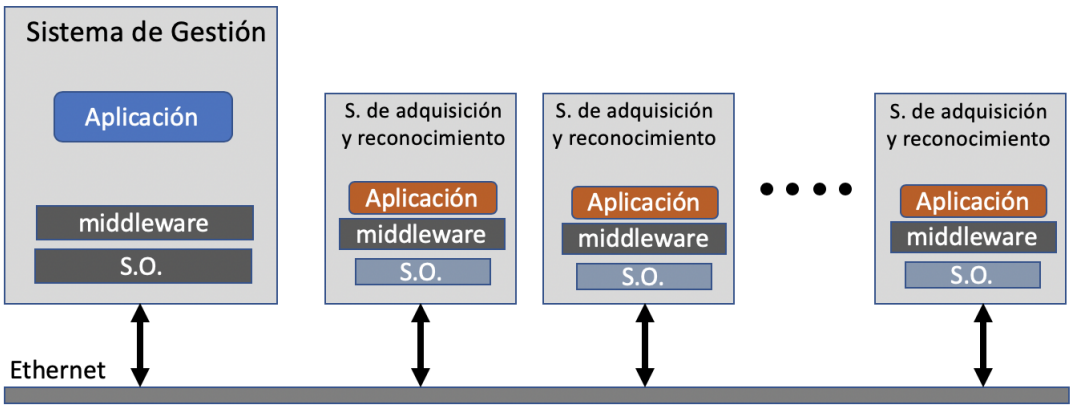


Figura 3.2: Esquema global del sistema

3.2 Casos de uso

A continuación se detallan algunos casos de uso de la aplicación. Estos casos de uso permiten identificar la secuencia de acciones que realizan los actores del proceso.

3.2.1. Caso de uso 1

En este caso de uso se identifican las acciones asociadas a una petición de conexión.

Referencia	CU-001
Descripción	Establecimiento de la conexión.
Actores	Usuario y Sistema de gestión.
Pre-condiciones	El sistema no está conectado al dispositivo de adquisición y reconocimiento.
Post-Condiciones	El sistema quedará conectado al dispositivo de adquisición y reconocimiento.
Secuencia de eventos	
1. El usuario selecciona una interfaz de red. 2. El usuario escribe el IPPort de la conexión. 5. El usuario pulsa el botón de Start.	 3. El SG comprueba los parámetros de la conexión. 4. El SG habilita el botón de Start de la interfaz 6. El SG realiza la conexión

Tabla 3.1: Caso de uso 1.

3.2.2. Caso de uso 2

Establecida la conexión, el usuario realiza una petición de adquisición de imagen.

Referencia	CU-002
Descripción	Adquisición de imagen
Actores	Usuario, SG y SAR
Pre-condiciones	El sistema dispone de una conexión entre SG y SAR
Post-Condiciones	La imagen es recibida por el SG.
Secuencia de eventos	
1. El usuario establece como topic COMMAND 2. El usuario establece como image topic IMAGE (valor por defecto) 3. El usuario establece como respuesta RESPONSE (valor por defecto) 4. El usuario pulsa el botón de FRAME	5. El SG valida los parámetros de la petición. 6. El SG trasmite a SAR la orden recibida a través del middleware 7. El SAR ejecuta el manejador de la orden 8. El SAR adquiere la imagen de la cámara 9. Si se ha habilitado la opción de clasificación, se realiza la clasificación del objeto 10. Si se ha habilitado la opción de recortar la imagen, se realiza el recorte de la imagen 11. El SAR publica la imagen en el middleware 12. El SG lee la imagen del middleware 13. El SG muestra la imagen en la interfaz gráfica Si la opción en la interfaz de mostrar imagen está habilitada, se muestra la imagen en la interfaz 14. El SG muestra el texto recibido desde el SAR en la consola si está habilitada la opción de *verbose* Si la opción en la interfaz de mostrar imagen está habilitada, se muestra la imagen en la interfaz

Tabla 3.2: Caso de uso 2.

3.2.3. Casos de uso 3, 4, 5 y 6

Los siguientes casos de uso son genéricos y detallan las acciones cuando se pulsan distintos botones de la interfaz como son *Do classification Clip image*, *Show image Panel*, *Show verbose console*.

El caso 3 se refiere a la selección de la opción para realizar la clasificación del objeto. El caso 4 detalla la selección del recorte de imagen. El caso 5 permite, si se habilita, mostrar la imagen adquirida en el panel de la interfaz dedicado a tal efecto. Por último, el caso 6 permite que se muestren por consola los mensajes enviados desde el SAR.

La descripción de todos estos casos sigue el mismo esquema y, por lo tanto, se detallan como un caso genérico en el que XX representa el número de caso y ZZ el botón a pulsar.

Referencia	CU-0XX
Descripción	Selección opción de imagen ZZ
Actores	Usuario, SG y SAR
Pre-condiciones	El sistema dispone de una conexión entre SG y SAR
Post-Condiciones	La opción seleccionada es trasmitida al SAR.
Secuencia de eventos	
1. El usuario pulsa el botón ZZ	2. El SG habilita o deshabilita el botón pulsado (ZZ) según su estado anterior 3. El SG trasmite a SAR la orden recibida a través del middleware 4. El SAR ejecuta el manejador de la orden 5. El SAR define el nuevo estado habilitando o deshabilitando la opción seleccionada

Tabla 3.3: Caso de uso 3, 4, 5 y 6.

3.2.4. Caso de uso 7

Establecida la conexión, el usuario realiza una petición de generación de un vídeo con las imágenes adquiridas.

Referencia	CU-007
Descripción	Adquisición de vídeo
Actores	Usuario, SG y SAR
Pre-condiciones	El sistema dispone de una conexión entre SG y SAR
Post-Condiciones	Un conjunto de imágenes permiten realizar el vídeo por el SG.
Secuencia de eventos	
1. El usuario pulsa el botón de VIDEO	
	2. El SG valida los parámetros de la petición. 3. El SG habilita el boton de Stop de la interfaz y desahilita los botones de Frame y video 4. El SG trasmite a SAR la orden recibida a través del middleware 5. El SAR ejecuta el manejador de la orden 6. El SAR adquiere la imagen de la cámara 7. Si se ha habilitado la opción de clasificación, se realiza la clasificación del objeto 8. Si se ha habilitado la opción de recortar la imagen, se realiza el recorte de la imagen 9. El SAR publica la imagen en el middleware 10. El SG lee la imagen del middleware repetición desde el punto 6, 7, 8, 9 y 10 11. El SG acumula las imágenes leídas
12. El Usuario pulsa el botón de STOP de la interfaz.	
	13. El SG genera un vídeo con las imágenes leídas

Tabla 3.4: Caso de uso 7.

3.2.5. Caso de uso 8

La interfaz bajo la acción del usuario guarda las imágenes en el disco.

Referencia	CU-008
Descripción	Guarda la imagen adquirida
Actores	Usuario, SG
Pre-condiciones	El sistema dispone de una conexión entre SG y SAR y se ha adquirido una imagen al menos
Post-Condiciones	Se guarda en el disco la imagen en la ruta y nombre especificado
Secuencia de eventos	
1. El usuario define la ruta en la que se guardarán las imágenes (hay definida una ruta por defecto) 2. El usuario pulsa el botón de *Set current directory*	
	3. el SG almacena la ruta definida
4. El usuario pulsa el botón de *Austo-save captured image* habilitando y deshabilitando si se pulsa de nuevo.	
	5. El SG guarda el estado del botón (habilitado o deshabilitado) y deshabilita o habilita el botón de *Save image*.
6. A cada imagen recibida según Caso 2, la imagen se guarda en la ruta y nombre con etiqueta adicional de la secuencialidad de la imagen si la opción de autoguardado está habilitada	
	7. Si está habilitado el botón de *Save image*, la última imagen adquirida se guarda en la ruta y nombre de archivo especificado.

Tabla 3.5: Caso de uso 8.

3.3 Requisitos de base del sistema

A continuación se detallan los requisitos iniciales para el desarrollo del TFG. La base para establecer los requisitos de base son los casos de uso descritos en la sección anterior.

Los requisitos se han estructurado en diferentes bloques atendiendo a criterios funcionales del sistema.

Los siguientes requisitos están orientados a esta visión global.

3.3.1. Subsistema de gestión

El subsistema de gestión se orientará a la interacción con el usuario a través de una interfaz gráfica y a enviar ordenes a los distintos sistemas de adquisición y reconocimiento para adquirir imágenes y realizar el proceso de reconocimiento. La figura 3.3 describe los componentes e interacciones del SG.

Por un lado dispondrá de una interfaz gráfica con la que un usuario podrá seleccionar el SAR a conectarse y una serie de opciones de comunicación para adquirir imágenes

Identificador	Requisito
RB-0010	El sistema estará formado por dos o más subsistemas: uno de gestión (SG) y uno o varios de adquisición y reconocimiento (SAR).
RB-0020	Los subsistemas conformarán un sistema distribuido conectados por red local.
RB-0030	Todos los subsistemas dispondrán de una capa de software middleware.
RB-0040	El middleware (multipeer) ofrecerá los servicios de conexión y/o desconexión entre subsistemas.
RB-0050	El middleware (multipeer) ofrecerá una lista de "topics"para identificar los objetos que serán utilizados en la comunicación entre subsistemas.
RB-0060	El middleware (multipeer) proporcionará los servicios para protocolos de publicación/subscripción a las aplicaciones.
RB-0070	El middleware (multipeer) proporcionará los mecanismos para ejecutar procedimientos locales cuando se realice una modificación de un objeto determinado.

Tabla 3.6: Requisitos de base

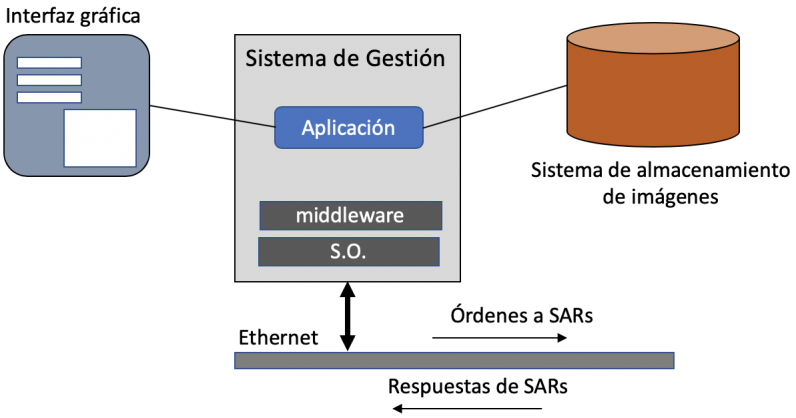


Figura 3.3: Esquema del subsistema de gestión SG.

o vídeos. En el proceso de adquisición de imágenes para la generación del sistema de reconocimiento, guardará las imágenes en el disco bajo una organización que identifique el nombre o etiqueta de la pieza y las imágenes adquiridas. Con el conjunto de imágenes etiquetadas aplicará el proceso de generación del modelo que, posteriormente, será trasladado a los SARs para que realicen el reconocimiento automático de piezas.

Los siguientes requisitos están orientados al subsistema de gestión.

3.3.2. Subsistema de generación del modelo de redes neuronales

El subsistema de generación del modelo de redes neuronales es el componente que a partir de la información de las imágenes organizadas en directorios con una etiqueta identificada aplica técnicas de aprendizaje profundo para generar una red neuronal multicapa que sea capaz de reconocer los objetos de las mismas características. La figura 3.4 describe los componentes e interacciones del SG.

La entrada al proceso de generación del modelo será la almacenada en el sistema de almacenamiento generada por el subsistema de gestión con la colaboración del SAR. La entrada será un conjunto de imágenes por etiqueta que alimentan a las librerías de

Identificador	Requisito
RB-0100	El subsistema de gestión dispondrá de una interfaz gráfica que permitirá interactuar con el usuario.
RB-0110	El subsistema de gestión enviará las órdenes introducidas por el usuario a través de la interfaz gráfica a los otros subsistemas y procesará la información de respuesta cuando la haya.
RB-0120	El subsistema de gestión (SG) podrá enviar la ordenes para conectarse y/o desconectarse con cualquier SAR.
RB-0130	El subsistema de gestión (SG) podrá enviar la petición al SAR seleccionado para adquirir una imagen y recibirla.
RB-0140	El subsistema de gestión (SG) podrá enviar la petición al SAR seleccionado para iniciar la adquisición de video y pararlo.
RB-0150	El subsistema de gestión (SG) etiquetará las imágenes recibidas con un prefijo dado y las almacenará en un directorio determinado del sistema de archivos.
RB-0160	El subsistema de gestión (SG) guardará todas las imágenes recibidas etiquetadas con una mismo etiqueta en un directorio con el mismo nombre.

Tabla 3.7: Requisitos del sistema de gestión

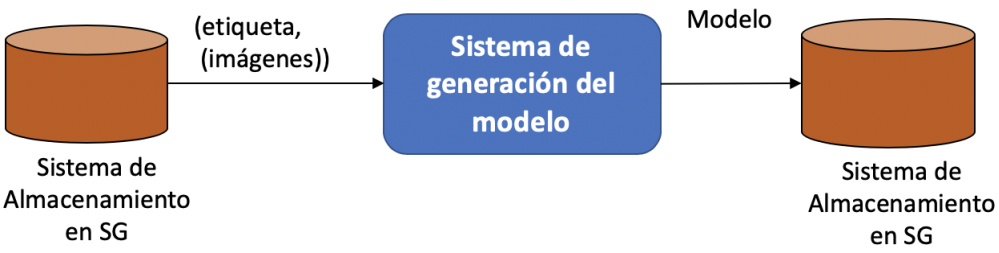


Figura 3.4: Esquema del subsistema de generación del modelo.

aprendizaje profundo. La salida será un modelo de red neuronal multicapa que se utilizará para el reconocimiento de los objetos en el SAR.

Los siguientes requisitos están orientados al subsistema de generación del modelo (SGM).

3.3.3. Subsistema de adquisición y reconocimiento

El subsistema de adquisición y reconocimiento (SAR) se diseñará para la interacción con la cámara para la adquisición de imágenes. El SAR recibirá ordenes desde el SG para que adquiera y envíe una imagen o generé un vídeo. Asimismo, podrá recibir un modelo con el que aplicar el reconocimiento de imágenes y devolver la imagen con el resultado del proceso de reconocimiento. La figura 3.5 describe los componentes e interacciones del SAR.

Identificador	Requisito
RB-0200	El subsistema SGM aplicará técnicas basadas en aprendizaje profundo para la generación del modelo.
RB-0210	El subsistema SGM leerá el conjunto de imágenes asociados a una etiqueta y alimentará con ellas la librería de aprendizaje profundo.
RB-0220	Una vez leídas todas la etiquetas con sus imágenes, el subsistema SGM invocará la generación del modelo a la librería.
RB-0230	El subsistema SGM guardará en el sistema de almacenamiento el modelo generado.
RB-0240	El subsistema SGM utilizará la librería ofrecida por las herramientas KERAS y Tensorflow para el proceso del aprendizaje y generación del modelo.
RB-0250	El subsistema SGM generará estadísticas sobre el proceso de generación para informar de los tiempos de generación, número de nodos, y otras características asociadas a este proceso.
RB-0260	El subsistema SGM se organizará como un proceso independiente de los otros subsistemas.
RB-0270	El subsistema SGM generará el modelo en un fichero con formato legible en JSON.

Tabla 3.8: Requisitos del subsistema de generación del modelo

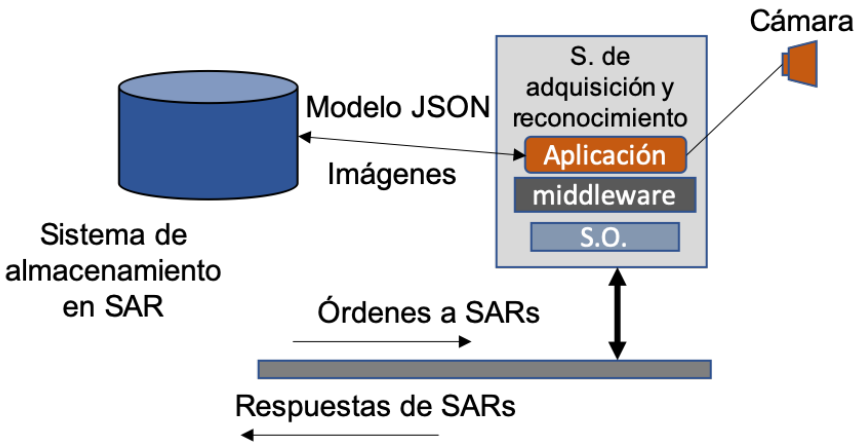


Figura 3.5: Esquema del subsistema de adquisición y reconocimiento SAR.

Los siguientes requisitos están orientados a sistema de adquisición y reconocimiento.

Identificador	Requisito
RB-0300	Cada SAR manejará una cámara para la adquisición de imágenes.
RB-0310	El SAR etiquetará la imagen con una marca temporal.
RB-0320	En repuesta a una petición del SG, el SAR adquirirá una imagen y la enviará al SG.
RB-0330	En repuesta a una petición del SG, el SAR adquirirá una imagen con un marco circunscrito a la pieza y la enviará al SG.
RB-0340	En repuesta a una petición del SG, el SAR adquirirá una secuencia de imágenes conformando un video e irá enviando la secuencia de imágenes al SG.
RB-0350	El SG podrá almacenar las imágenes etiquetadas en diferentes subdirectorios según el tipo de pieza.
RB-0360	El SG podrá generar un modelo a partir del análisis de todas las imágenes mediante técnicas basadas en redes neuronales.
RB-0370	El modelo generado por el SG será utilizado por los SAR para realizar la identificación del tipo de pieza adquirida.
RB-0380	El SAR realizará, bajo petición del SG, la identificación de la pieza de acuerdo al modelo y devolverá el resultado.
RB-0390	El resultado devuelto por el SAR estará formado por un vector de probabilidades de pertenencia de la imagen adquirida a las clases del modelo.
RB-0400	El resultado devuelto por el SAR incluye la clase del modelo con mayor probabilidad.

Tabla 3.9: Requisitos del sistema de adquisición y reconocimiento

3.3.4. Requisitos de la Interfaz de usuario

La interfaz de usuario la implementará el SG y en ella se visualizarán las distintas opciones que puede elegir el usuario para operar con los sistemas de adquisición y reconocimiento de piezas.

Los siguientes requisitos están orientados a la funcionalidad a ofrecer en la interfaz de usuario.

Identificador	Requisito
RB-0500	La interfaz de usuario será lanzada al ejecutar el SG.
RB-0510	La interfaz de usuario permitirá seleccionar la interfaz de red (IP) del SAR con el que se desea interactuar.
RB-0520	La interfaz de usuario permitirá definir la interfaz de red y el puerto del grupo asociado al middleware.
RB-0530	La interfaz de usuario permitirá seleccionar el tipo de orden a enviar al SAR seleccionado (Command Topic).
RB-0540	La interfaz de usuario permitirá seleccionar el tipo de imagen a recibir del SAR seleccionado (Image Topic).
RB-0550	La interfaz de usuario permitirá seleccionar el tipo de respuesta que se espera recibir del SAR seleccionado (Response Topic).
RB-0560	La interfaz de usuario podrá seleccionar entre recibir una imagen o un video. En el caso del video se controlará el inicio y fin del mismo.
RB-0570	La interfaz de usuario permitirá que en la respuesta de la imagen se realice el proceso de reconocimiento en el SAR y se informe del resultado.
RB-0580	La interfaz de usuario podrá solicitar que las imágenes se circunscriban estrictamente a la imagen capturada.
RB-0590	La interfaz de usuario podrá solicitar que las imágenes se visualicen en un marco para tal efecto en la interfaz.
RB-0600	La interfaz de usuario dispondrá de una consola por la que se mostrarán los mensajes de texto recibidos del SAR. La visualización se podrá activar y desactivar por el usuario
RB-0610	La interfaz de usuario dispondrá de un marco en el que mostrar las imágenes.

Tabla 3.10: Requisitos de la interfaz de usuario

3.3.5. Requisitos no funcionales del Subsistema de Gestión (SG)

Los siguientes requisitos no funcionales hacen referencia a las tecnologías a utilizar para el proceso de análisis de imágenes en el SG y la generación de modelos para el sistema SAR. Asimismo, se detallan las opciones de lenguajes a utilizar en desarrollo tanto del SG como del SAR.

Identificador	Requisito
RB-0700	El análisis de las imágenes para la generación del modelo se realizará mediante redes neuronales.
RB-0710	Para el análisis de las imágenes y la generación de los modelos de redes neuronales se utilizarán los paquetes Keras y Tensorflow.
RB-0720	Con el fin de comparar prestaciones se realizarán dos versiones de la generación de modelos con los lenguajes R y Python como capas por encima de los paquetes de imágenes.
RB-0730	La aplicación desarrollada en el SG para el análisis de imágenes y generación de modelos sera multiplataforma.
RB-0740	La interfaz gráfica de la aplicación se realizará en Java.
RB-0750	Se podrán definir distintas tallas de clases con el fin de comparar niveles de acierto entre diferentes modelos.

Tabla 3.11: Requisitos del análisis de imágenes

3.3.6. Requisitos no funcionales del Subsistema de adquisición y reconocimiento (SAR)

Los siguientes requisitos no funcionales hacen referencia a la vista física de despliegue del sistema global.

Identificador	Requisito
RB-0800	Cada SAR se implementará mediante un sistema empotrado dedicado.
RB-0810	En el prototipo de este TFG se utilizará la RaspberryPi 4 para la implementación del SAR.
RB-0820	En el prototipo de este TFG se incluirá una cámara RaspiCam.
RB-0830	El sistema operativo del SAR será Raspbian basado en una distribución de Linux basada en Debian.
RB-0840	El desarrollo de los programas en el SAR se realizará en C++.
RB-0850	El software del SG será multiplataforma.
RB-0860	La interfaz de usuario del SG se realizará en Java.
RB-0870	El SG deberá poder ejecutar lenguajes interpretados como R, Java y Python.

Tabla 3.12: Requisitos de despliegue

3.4 Conclusiones

En este capítulo se ha abordado la especificación del sistema a desarrollar. Pare ello, se ha planteado inicialmente un esquema global sobre el que se han definido un conjunto de casos de uso del sistema. Estos casos de usos permiten definir la forma en la que se va a usar el sistema y la reacción del mismo ante las opciones de uso. Los casos de uso han permitido desarrollar el conjunto de requisitos de base que son la base para abordar el diseño e implementación de las aplicaciones. En los requisitos se ha diferenciado entre funcionales y no funcionales focalizándose para cada subsistema del sistema final.

CAPÍTULO 4
Diseño de la solución

A partir de los requisitos identificados en el capítulo 3, se plantea a continuación el diseño del sistema. Tal y como se ha descrito, la aplicación global incluye dos subsistemas de ejecución distintos: computador de gestión y sistema empotrado de adquisición y reconocimiento de imágenes. En este capítulo se realiza el diseño global del sistema por subsistemas y el detallado de aquellos servicios relevantes de las aplicaciones.

4.1 Arquitectura del Sistema Global

La arquitectura del sistema global se representa en la figura 4.1. En ella se muestran los dos subsistemas que configuran el sistema completo.

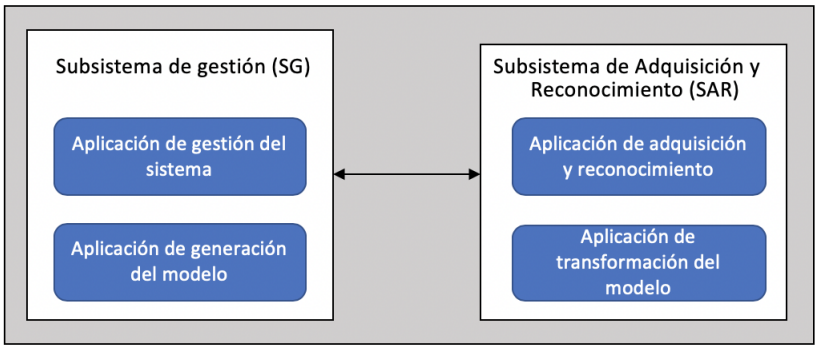


Figura 4.1: Visión global del sistema

Por un lado se identifican las aplicaciones que se ejecutarán en el sistema de gestión (SG) y aquellas que se ejecutarán en el sistema empotrado de adquisición y reconocimiento (SAR). La vista que se presenta en la figura 4.1 corresponde a la vista de despliegue del diseño. En ella se identifican las aplicaciones que se ejecutarán en cada una de las plataformas de ejecución: el sistema de gestión que se ejecutará en la máquina en la que el usuario interactúa con la interfaz de usuario y la máquina dedicada a la adquisición y reconocimiento de imágenes o sistema empotrado.

A continuación se realiza el diseño de cada una de las aplicaciones que conforman la vista presentada.

4.2 Aplicación de gestión del sistema

La aplicación de gestión del sistema realizará las funciones de interacción con el usuario y, a partir de las órdenes introducidas, se comunicará con la aplicación de adquisición y reconocimiento de imágenes.

La figura 4.2 muestra los componentes internos de la aplicación.

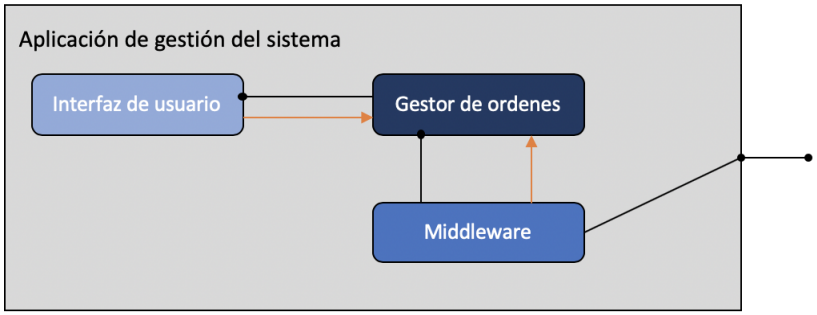


Figura 4.2: Componentes del SC

La aplicación del sistema de gestión está compuesta por tres componentes identificados en la figura.

- Gestor de órdenes: Es el módulo principal de la aplicación y se encarga de lanzar e interactuar con el resto de componentes. La línea negra en la figura representa el uso de los servicios de los componentes por este módulo. Por otro lado, la respuesta de algunos servicios se realiza mediante la generación de un evento o una llamada (*callback*) del otro módulo que se representa mediante la flecha naranja.

- Interfaz de usuario: es el componente encargado de gestionar la interfaz mediante objetos gráficos (texto, botones, paneles, etc.) en la que el usuario puede incidir. Las acciones de respuesta a botones que el usuario genera son enviadas al gestor de órdenes para que realice las operaciones oportunas.

- Middleware: es el componente que ofrece un conjunto de servicios para la comunicación y distribución de la información entre las distintas aplicaciones ubicadas en distintos nodos de la red. Tiene una fuerte interacción con el sistema operativo que los manejadores de dispositivos de red. Permite de una manera eficiente comunicar la aplicación de la gestión del sistema con el resto de componentes SAR que incorporan este mismo componente. Los mensajes desde otros componentes que requieren la intervención del módulo de gestor de órdenes son gestionado mediante *callbacks* hacia dicho módulo. Este módulo es usado por dicho módulo en aquellas operaciones iniciadas por la interfaz.

Un mayor detalle de los componentes del SG se detalla en la figura 4.3. En ella se detallan las principales clases de paneles usadas en la interfaz de usuario y las librerías de clases básicas proporcionados por las clases de Java. Así mismo, se descompone el componente middleware en dos módulos encargados de la gestión de los mensajes por la red y la gestión del los objetos compartidos. Estos componentes se describirán con un mayor detalle en las secciones siguientes.

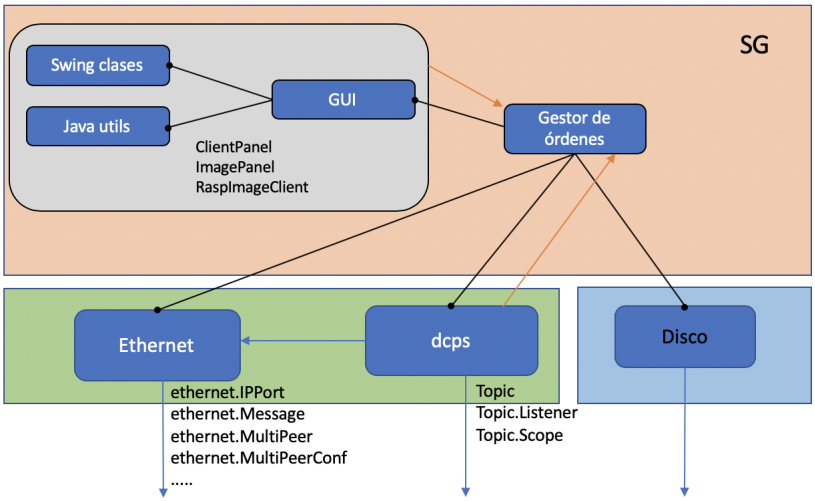


Figura 4.3: Detalles de los componentes del SG

4.2.1. Gestor de órdenes

Este componente es el módulo principal de la aplicación y se encarga de iniciar y gestionar las interacciones con la interfaz de usuario y el middleware. La figura 4.20 detalla mediante un diagrama de estado su funcionamiento.

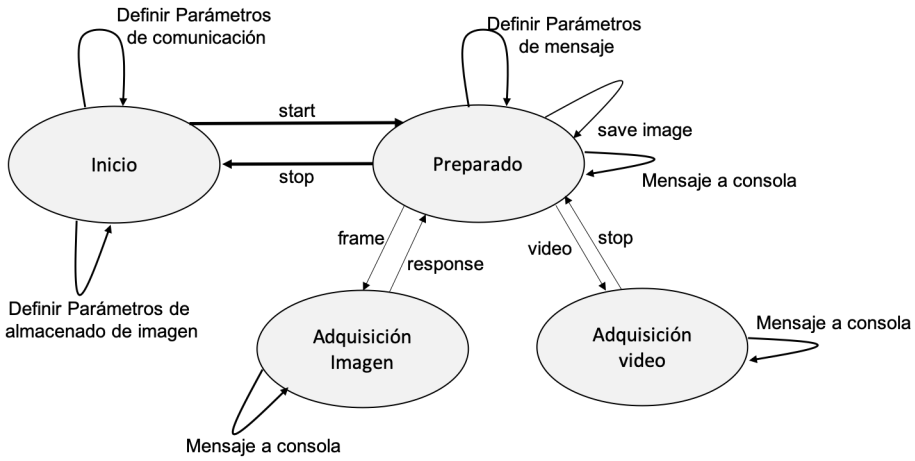


Figura 4.4: Diagrama de estado del Gestor de órdenes

Los estados del componente se describen a continuación:

Inicio : En este estado, el componente crea la interfaz de usuario definiendo los distintos campos mediante la configuración por defecto. En este estado se aceptarán distintos cambios de los campos de la configuración por defecto. De este estado se saldrá hacia el estado Preparado cuando el usuario establezca la comunicación con un dispositivo seleccionado y tenga una respuesta positiva a la conexión.
A este estado se vuelve desde el estado Preparado cuando en la interfaz se pulse el botón correspondiente a parar la conexión.

Preparado : En este estado, la aplicación está conectada a un dispositivo de adquisición y reconocimiento de imagen. Este estado admite que se puedan cambiar los parámetros asociados al mensaje a enviar (tipo de tópico, tipo de respuesta, que realice un reconocimiento cuando se pida la imagen, etc.), guardar la imagen o vídeo ad-

quirido y volcar en la consola los mensajes recibidos. Del estado Preparado se sale a los estados de:

- Adquisición de imagen: cuando su pulsa el botón de frame y se vuelve cuando se recibe el mensaje con la imagen
- Adquisición de vídeo: cuando su pulsa el botón de vídeo y se vuelve cuando se pulsa el botón de stop.

Adquisión imagen : En este estado se está esperando que llegue el mensaje del sistema de adquisición con la imagen. Cuando se reciba ésta, se volverá al estado Preparado.

Adquisión vídeo : En este estado se está esperando que lleguen los mensaje del sistema de adquisición con las imágenes que configurarán el vídeo. Cuando se pulse el botón de stop, se volverá al estado Preparado.

4.2.2. Interfaz de usuario

Para la interfaz del usuario se ha utilizado el Java Swing por su versatilidad e integrabilidad en Java.

La interfaz de usuario se estructura en dos vistas orientados cada uno de ellos a una funcionalidad. Por un lado la vista 1 ofrece los elementos gráficos para el control de la aplicación. En ella, el usuario, podrá iniciar la conexión un un determinado SAR, iniciar la captura de imágenes , arrancar y parar la generación de un vídeo y elegir las distintas opciones de la captura.

La figura 4.5 muestra el diseño gráfico de la interfaz y detalla los objetos gráficos que se utilizan.

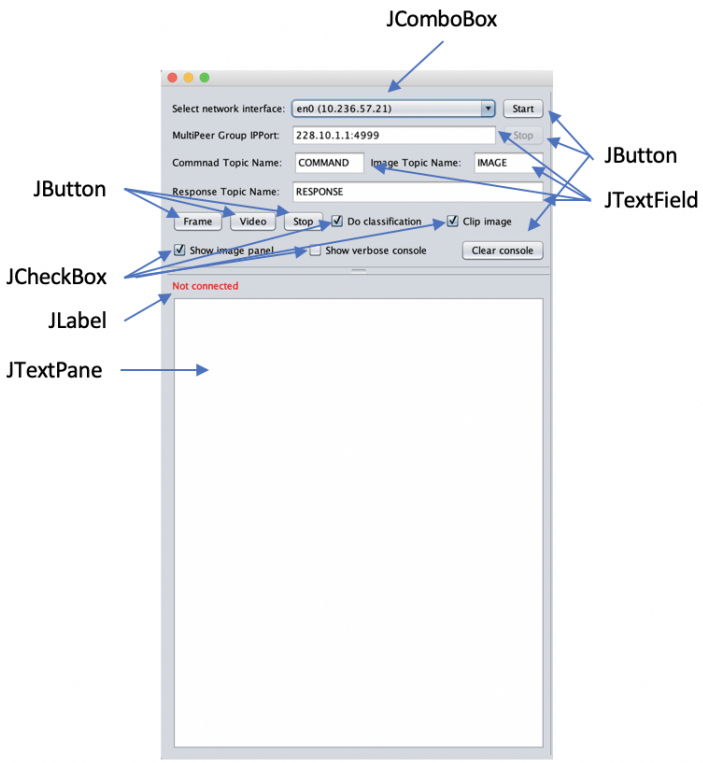


Figura 4.5: Vista 1: control de la aplicación

En el diseño de la vista 1 se han utilizado las siguientes clases definidas en la librería *Swing* de Java para implementar la interfaz de control.

- JFrame: componente básico utilizado para implementar una interfaz visual con la libraría *Swing*. Esta clase encapsula una ventana clásica de cualquier sistema operativo con entorno gráfico.
- JComboBox: permite seleccionar un *String* de una lista.
- JButton: muestra un botón.
- JTextField: permite al operador del programa ingresar una cadena de caracteres por teclado.
- JCheckBox: permite implementar un cuadro de selección.
- JTextPane: permite ingresar múltiples líneas de caracteres.
- JLabel: permite mostrar texto.
- JFileChooser: permite elegir el *path* del almacenamiento.
- JList: permite crear una lista en una zona de texto.
- JScrollPane: Este componente nos proporciona las barras de desplazamiento vertical y horizontal por si lo que queremos mostrar es muy grande podernos mover y ver todo el contenido.

A continuación se detalla la información que cada uno de los campos definidos en la interfaz representa en el contexto de la aplicación.

- Select network interface: identifica la dirección IP en la que el SAR está ubicado.
- MultiPeer Group IPPort:
- Command Topic Name: el nombre del *topic* por el que se comunica el SG con el SAR para enviar/recibir comandos. Esto creará un *scope* por el que se enviará la información y añadirá un *listener*.
- Image Topic Name: el nombre del *topic* por el que se comunica el SG con el SAR para enviar/recibir imagenes. Esto creará un *scope* por el que se enviará la información y añadirá un listener.
- Response Topic Name: el nombre del *topic* por el que se comunica el SG con el SAR para enviar/recibir resupesta. Esto creará un *scope* por el que se enviará la información y añadirá un *listener*.
- Frame, Video y Stop: son los botones para mostrar una imagen, vídeo o para el vídeo.
- Do classification: marca la opción de hacer o no hacer clasificación de las imágenes recibidas. Es decir, si mostrar el resultado obtenido de realizar la clasificación o no.
- Clip image: marca la opción de recortar la imagen en una imagen más pequeña mostrando sólo la pieza o ver la imagen completa.
- Show image panel: Marca la opción de mostrar o no mostrar la segunda parte de la interfaz 4.6.
- Show verbose console: esta opción se utiliza para ver más información sobre todos los procedimientos de comunicación, envío, etc...

- Clear console: se utiliza para eliminar todo lo escrito en la consola.
- Connected/Not connected: muestra si estamos conectados o no a algún SAR.

En la figura 4.6 se representa la segunda vista de la interfaz. Al igual que en el caso anterior, se detallan en la figura las clases de objetos que se utilizan para estructurar la información que han sido descritos en la lista de clases de *Swing* usadas.

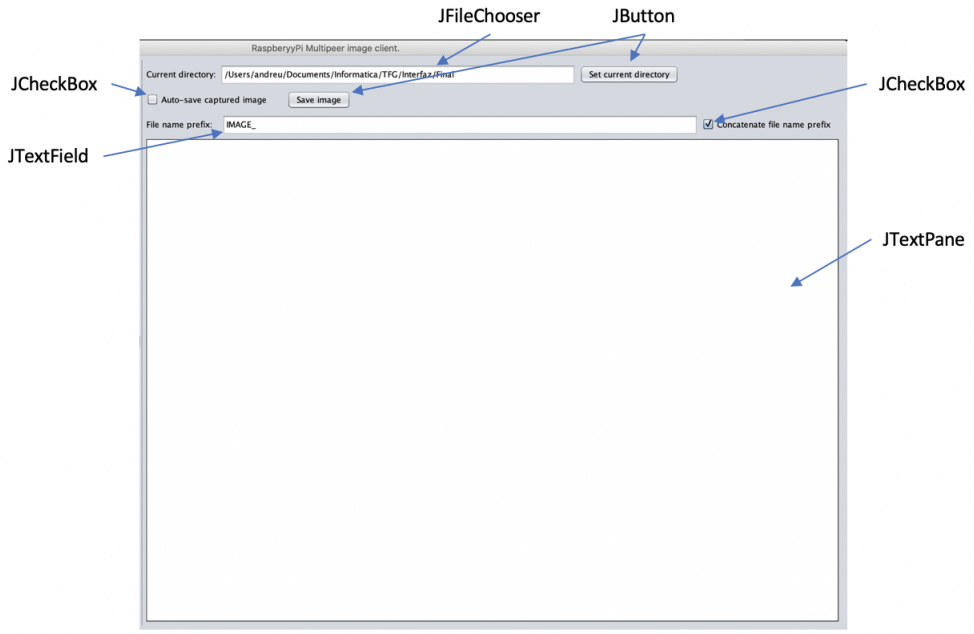


Figura 4.6: Vista 2 de la interfaz

La información que se representa en la vista 2 es la siguiente:

- Current directory: muestra el *path* de la carpeta que utilizaremos para guardar las imágenes que obtengamos con el SG.
- Set current directory: botón para elegir el directorio.
- Auto-save captured image: si la opción está marcada no tendremos que apretar el botón de *save image* para guardar la imagen en el directorio elegido, si no que la guardará automáticamente.
- Save image: botón que da la orden de guardar la imagen que se muestra en *pane*.
- File name prefix: prefijo que le podemos dar a la imagen a la hora de guardarla.
- Concatenate file name prefix: marca la opción de concatenar el prefijo o no.
- ImagePane: panel donde nos muestra la imagen obtenida o el vídeo.

4.3 Aplicación de generación del modelo

En el prototipo desarrollado en este TFG, la generación del modelo se ha diseñado como una aplicación independiente de las otras aplicaciones para ser lanzada fuera de línea. Aunque su diseño integrado en la aplicación anterior sería una opción a implementar en un futuro, la independencia de la generación del modelo permite su lanzamiento con

distintas opciones de entrenamiento con el fin de experimentar y ajustar el modelo para una mejor sintonización. Una vez se deciden los parámetros finales, sería más adecuado su integración. Esta acción se ha dejado como trabajo futuro.

El proceso de generación del modelo de reconocimiento se representa en la figura 4.7. En ella se pueden identificar los distintos procedimientos que se realizan sobre las imágenes para conseguir la red neuronal completa para el reconocimiento de piezas. El esquema detalla el tratamiento de una imagen que se repetiría para cada una de las imágenes que conforman el conjunto de imágenes adquiridas. La salida obtenida en la fase final es incremental y va incorporando los distintos conjuntos de imágenes de cada una de las piezas a reconocer.

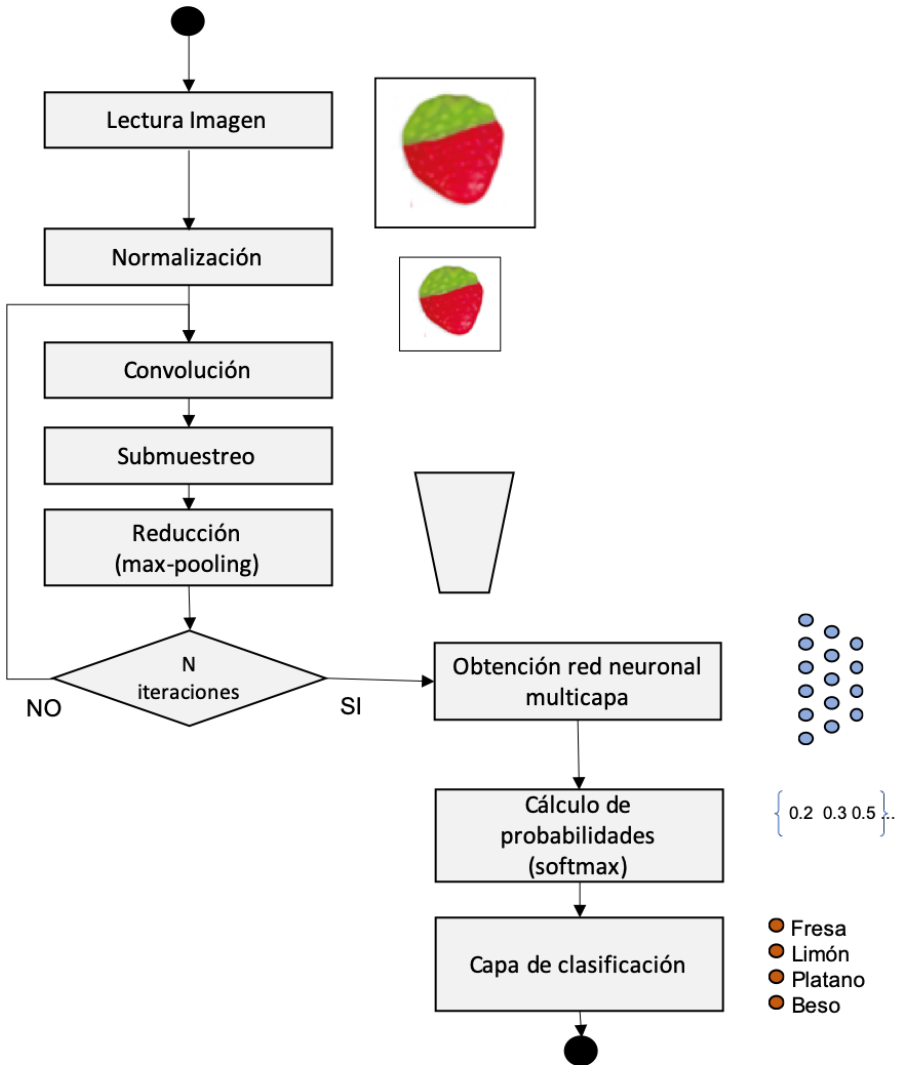


Figura 4.7: Flujograma del proceso de reconocimiento

A continuación se describe cada uno de los pasos del proceso.

Las redes neuronales convolucionales (RNC) que se han utilizado para el aprendizaje automático utilizan el aprendizaje supervisado para procesar sus capas con el fin de identificar distintas características en las entradas. Para ello, la RNC tiene varias capas ocultas especializadas organizadas en una jerarquía donde las primeras capas se pueden detectar curvas, lineas y especializándose conforme avanza el proceso hasta que llegar a capas más profundas donde se puede llegar a reconocer formas complejas como siluetas o rostros.

La red neuronal debe aprender a reconocer un número de objetos en el conjunto de imágenes. Para ello necesitaremos una gran cantidad de imágenes (200 imágenes de cada clase en nuestro caso). Esto es necesario para que la red pueda detectar las características únicas de cada objeto y a su vez, poder generalizarlo, ya que aunque sean objetos de la misma clase, ningún objeto es exactamente igual y podrán estar en diferentes posiciones a la hora de realizar el reconocimiento.

4.3.1. Proceso de aprendizaje

La red tiene como entrada los píxeles de una imagen. En este caso, las imágenes se restringen a 64x64 píxeles de alto y ancho, por lo que habrán $64x64 = 4{,}096$ neuronas. Además de esto, como las imágenes son en color se necesitaran 3 canales: rojo, verde y azul (RGB). Por lo tanto, el número de neuronas totales de entrada serán $64x64x3 = 122{,}88$. Estas neuronas configurarán la capa de entrada de la red.

Pre-procesamiento

Antes de alimentar la red con las imágenes se tienen que realizar un pre-procesamiento. Para ello deberemos normalizar los valores. Los colores tienen unos valores que van de 0 a 255, se realizará una transformación de cada píxel para que quede un valor entre 0.0 y 1.0. Esta transformación es el resultado de: $valor/255$.

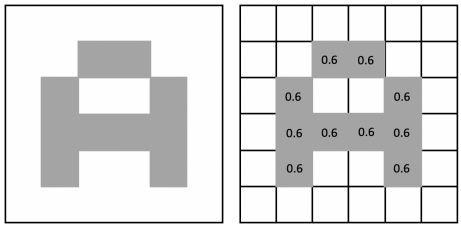


Figura 4.8: Letra en BN y su equivalente tras la transformación

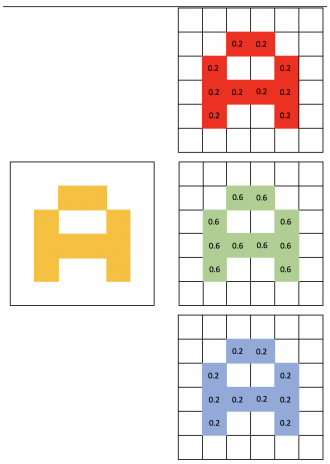


Figura 4.9: Letra en RGB y su equivalente tras la transformación

Convoluciones

Tras el pre-procesado empieza el procesos distintivo de las RNC. Se realizan llamadas “convoluciones” que toman "grupos de píxeles cercanos"de la imagen de entrada y van

operando matemáticamente (producto escalar) contra una matriz que se llama *kernel*. Ese *kernel* recorre todas las neuronas de entrada (de izquierda a derecha y de arriba a abajo) y genera una nueva matriz de salida que pasará a ser la nueva capa de neuronas ocultas. En un principio el *kernel* toma valores aleatorios que se irán ajustando mediante *backpropagation*. Al ser imágenes a color, el *kernel* será tridimensional, es decir, suponiendo que el *kernel* fuera igual a 3, sería 3x3x3 píxeles. Realmente sería un filtro con 3 *kernels* de 3x3 y luego esos 3 filtros se suman conformando una salida.

Filtro

No sólo se aplica un *kernel*, sino que se tendrán muchos *kernel* (el conjunto de *kernels* se llama filtros). Viendo el ejemplo de las figuras 4.10, 4.11 y 4.12, se tendrían 32 filtros, por lo que se obtendrían 32 matrices de salida. Al conjunto de las matrices de salida se le identifica como el mapa de características *feature mapping*. Cada matriz de salida será de 28x28x1 dando un total de 25.088 neuronas que correspondería a la primera capa oculta. Mientras se va desplazando el *kernel*, se van obteniendo nuevas imágenes filtradas por el *kernel*. En la primera convolución es como si se generan 32 imágenes filtradas. Estas nuevas imágenes están dibujando las características de la imagen original.

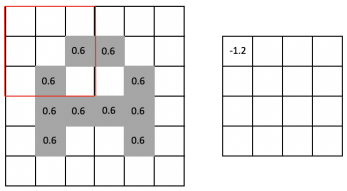


Figura 4.10: Primer paso del kernel en una convolución

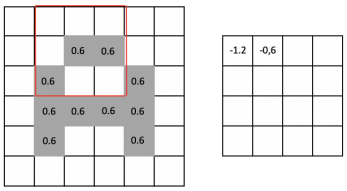


Figura 4.11: Segundo paso del kernel en una convolución

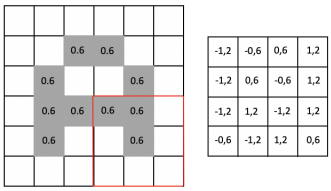


Figura 4.12: Tercer paso del kernel en una convolución

Función de activación

La más utilizada para este tipo de redes neuronales es la *Rectifier Linear Unit* (ReLu) y consiste en lo siguiente: $f(x) = max(0, x)$. Tras aplicar ReLu, se obtiene un mapa de detección de características.

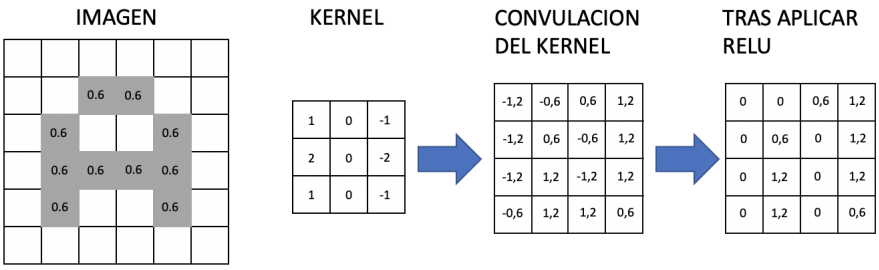


Figura 4.13: Proceso ReLu

Sub-muestreo

En este momento siguiendo el ejemplo anterior tendríamos una capa oculta de 25088, por lo que es necesario reducir el número de neuronas antes de hacer la siguiente convolución para evitar que se dispare el número de neuronas. Para conseguir reducir la cantidad de neuronas se hace un proceso de sub-muestreo (*subsampling*). En este proceso se reducirá el tamaño de las imágenes filtradas pero deberán prevalecer las características más importantes detectadas en cada filtrado. El tipo de sub-muestreo más utilizado es el *Max-Pooling*. También es el que se utiliza en este trabajo

Fijándonos en el ejemplo de la figura 4.14. En este caso haríamos un *Max-pooling* de 2x2. Con esto recorreremos las 32 imágenes de características obtenidas de 28x28 *pixels* de izquierda a derecha y de arriba a abajo pero en vez de coger de a 1 *pixel*, cogeremos de 2x2, es decir, 2 de ancho por dos de alto, obteniendo 3 *pixeles*. En el caso del ejemplo se reduce a la mitad la imagen obtenida como resultado quedando 14x14. Por lo que obtendremos 32 imágenes de 14x14 reduciendo considerablemente el número de neuronas. Además la información más importante para detectar características sigue almacenada.

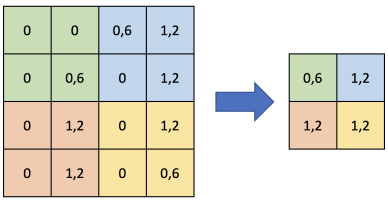


Figura 4.14: Subsampling con Max-Pooling

Convoluciones

La primera convolución 4.15 consiste en una entrada, unos filtros, se genera un mapa de características y se hace un sub-muestreo. Con este proceso se obtiene lo mostrado en la tabla 4.1.

Imagen de entrada	Aplicación del kernel	Obtención de feature mapping	Aplicación Max-Pooling	Salida de la convolución
28x28x1	32 filtros 3x3	28x28x32	2x2	14x14x32

Tabla 4.1: Primera convolución

La primera convolución se centra en detectar características primitivas como son las lineas o las curvas. Cuantas más capas de convolución, los mapas de características serán capaces de reconocer formas más complejas. En la segunda convolución se obtiene, tal y como se indica en la figura 4.16, un mapa de 7x7x64 a partir de una entrada de 14x14x32.

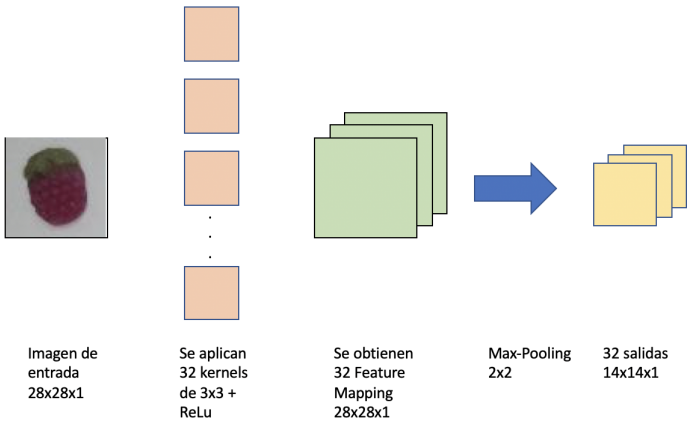


Figura 4.15: Primera convolución

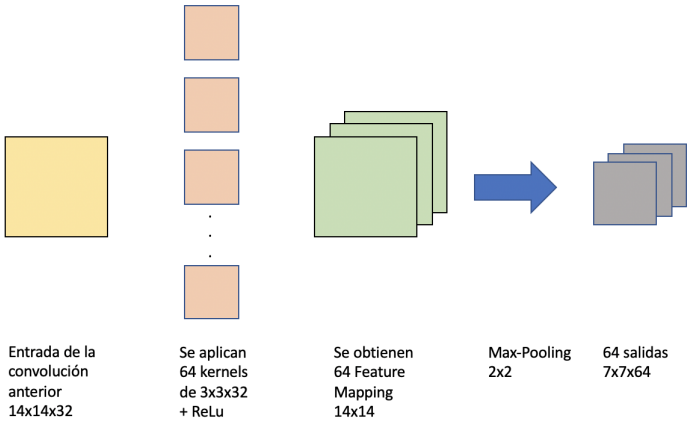


Figura 4.16: Segunda convolución y sucesivas

La tercera convolución empezará con un tamaño de 7x7 píxeles y después del *max-pooling* quedará en 3x3 por lo que sólo se podría hacer una convolución más. En esta explicación se ha partido de una entrada de 28x28 y se necesitan 3 convoluciones. Las imágenes utilizadas en el TFG son de 64x64 y requerirían 5 convoluciones.

Conexión con la red neuronal

Para terminar, utilizando la última capa oculta a la que se le hizo *subsampling*, la cual es "tridimensional"por tener alto, ancho y mapas (en nuestro ejemplo 3x3x128), la aplanamos. Al aplanarla deja de ser tridimensional y pasa a ser una capa de neuronas "tradicionales".

Entonces a esta nueva capa oculta "tradicional", se le aplica una función de clasificación múltiple conocida como *softmax* que conecta con la capa de salida final. Esta tendrá la cantidad de neuronas que corresponda con las clases que estamos clasificando. En el caso de la figura 4.17 serán 3 neuronas, ya que estamos clasificando entre perros, tortugas y gatos.

Las salidas tendrán un formato llamado *one-hot-encoding* en donde los resultados serían [1,0,0], [0,1,0] o [0,0,1] dependiendo de la clase.

La función *softmax* se encarga de pasar a probabilidad (entre 0 y 1) a las neuronas de salida. Por ejemplo [0.0, 0.7, 0.2, 0.1] nos indica que hay 0 % de probabilidades de que sea

Imagen de entrada	Aplicación del kernel	Obtención de feature mapping	Aplicación Max-Pooling	Salida de la convolución
14x14x32	64 filtros 3x3	14x14x64	2x2	7x7x64

Tabla 4.2: Segunda convolución

Imagen de entrada	Aplicación del kernel	Obtención de feature mapping	Aplicación Max-Pooling	Salida de la convolución
7x7x64	128 filtros 3x3	7x7x128	2x2	3x3x128

Tabla 4.3: Tercera convolución

una cola, 70 % de que sea una fresa y 20 % de que sea un plátano y 0.1 % de que sea un beso.

Ajuste de la red

Es el proceso por el que la RNC aprende de las imágenes analizadas. Después de la compilación la red se puede ajustar adaptando los pesos a un conjunto de datos dados de partida. El ajuste de la red requiere que se especifiquen los datos de partida, tanto una matriz de patrones de entrada, como de salida coincidentes. La red se forma aplicando el algoritmo de transmisión hacia atrás o retro-propagación *backpropagation* optimizándose para una función de pérdida especificados al compilar el modelo.

El algoritmo de retropropagación requiere que la red sea entrenada para un número específico de exposiciones llamadas épocas, al conjunto de las imágenes de entrenamiento. A partir de la entrada que se ha aplicado y de la salida generada, ésta se compara con la salida esperada determinando el error obtenido. Los errores se transmiten hacia atrás *backpropagation* con el fin de mejorar el valor de los pesos de las interconexiones entre capas de neuronas. A medida que se va iterando, los pesos de la red neuronal se ajustan hasta ser óptimos.

4.3.2. Modelo generado

El modelo generado determina la arquitectura de la red neuronal definido por el número de nodos, capas y pesos. Los modelos se definen instanciando las capas y conectándolas directamente entre sí en pares, para terminar especificando las capas que actuarán como entrada y salida del modelo.

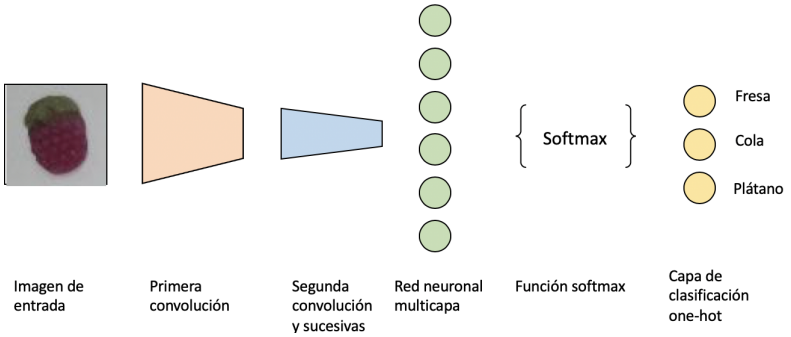


Figura 4.17: Arquitectura de una RNC

El modelo se utiliza para realizar el reconocimiento de las imágenes. El modelo se guarda en un fichero para ser enviado al SAR para que lo utilice en la fase de reconocimiento. El modelo, y por lo tanto el fichero, contiene: el modelo de la arquitectura generada y los pesos calculados durante el entrenamiento.

4.4 Aplicación de adquisición y reconocimiento

La aplicación de adquisición y reconocimiento está ubicada en el subsistema de adquisición y reconocimiento (SAR) y se encarga de realizar la adquisición de imágenes desde la cámara y aplicar el modelo de reconocimiento a las imágenes para su entrega a la aplicación de gestión del sistema ubicada en el SG.

La figura 4.18 muestra el diseño de primer nivel de la aplicación. En ella se identifican los principales componentes que la conforman.

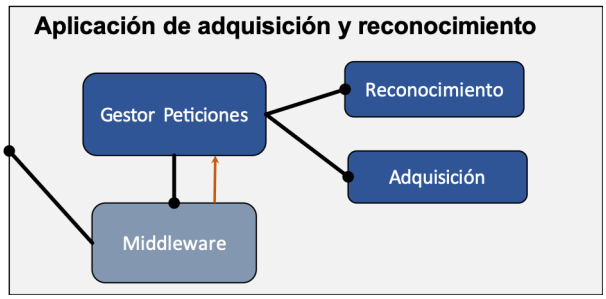


Figura 4.18: Subsistema de adquisición y reconocimiento

Los componentes en los que se descompone son:

- Gestor de órdenes: Es el componente principal y se encarga de recibir órdenes desde el SG y ejecutarlas. Las órdenes hacen referencia a la adquisición de imágenes y al reconocimiento utilizando el modelo generado en el proceso fuera de línea descrito en la aplicación de generación de modelo.
- Adquisición: este componente se corresponden con el manejador de la cámara y permite adquirir imágenes y realizar algunas operaciones básicas con ellas.
- Reconocimiento: cuando se pide desde el SG la adquisición con reconocimiento, este componente se encarga de aplicar la imagen al modelo de red neuronal generado previamente devolviendo la respuesta de ésta con el conjunto de probabilidades de cada clase de pieza.
- Middleware: es el componente que ofrece un conjunto de servicios para la comunicación y distribución de la información entre las distintas aplicaciones ubicadas en distintos nodos de la red. Tiene una fuerte interacción con el sistema operativo que los manejadores de dispositivos de red. Permite de una manera eficiente comunicar las aplicaciones en los dos nodos de ejecución. Los mensajes desde otros componentes que requieren la intervención del módulo de gestor de órdenes son gestionado mediante *callbacks* hacia dicho módulo. Este middleware tiene la misma interfaz y funcionalidad que el descrito en 4.2 aunque la implementación se realice en otro lenguaje.

Un mayor detalle de los componentes del SAR se detalla en la figura 4.19.Al igual que en el SG, el middleware se descompone en dos módulos encargados de la gestión de

los mensajes por la red y la gestión del los objetos compartidos. Estos componentes se describirán con un mayor detalle en las secciones siguientes.

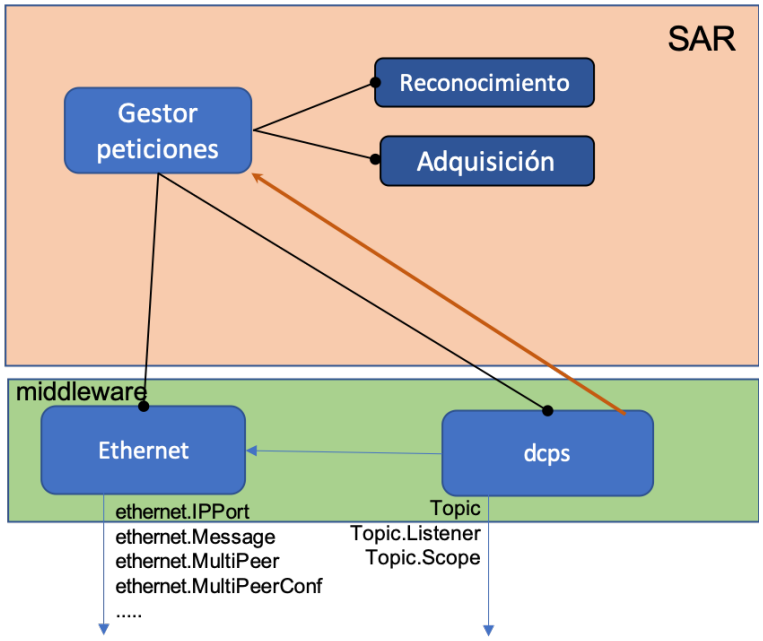


Figura 4.19: Arquitectura del SAR

4.4.1. Gestor de peticiones

Este componente es el módulo principal de la aplicación del Sistema de adquisición y reconocimiento (SAR) y se encarga de iniciar y gestionar las interacciones con la información y peticiones generadas desde el SG a través del middleware. La figura 4.20 detalla mediante un diagrama de estado su funcionamiento.

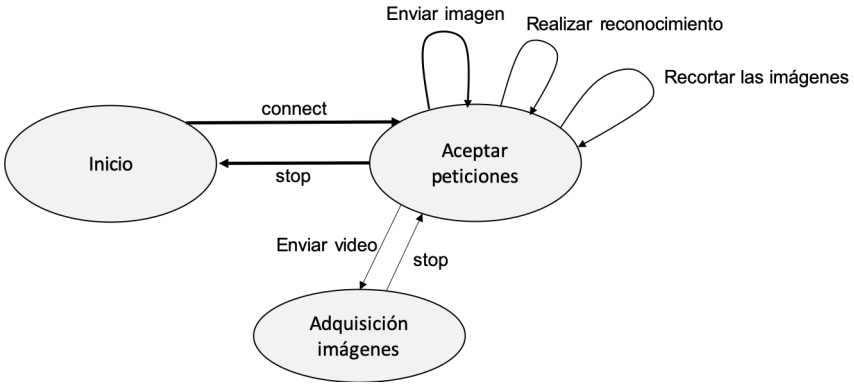


Figura 4.20: Diagrama de estado del Gestor de Peticiones

Los estados del componente se describen a continuación.

Inicio : En este estado, el componente inicializa los parámetros de configuración y espera que se establezca la conexión desde el SG. Una vez establecida la conexión, el gestor de peticiones cambia su estado para aceptar peticiones desde el SG hasta que se realiza la desconexión y se vuelva al estado inicial.

Aceptar peticiones : En este estado, la aplicación está conectada como dispositivo de adquisición y reconocimiento de imagen. En este estado puede recibir órdenes o peti-

ciones para la adquisición de una imagen, que se realice el reconocimiento de éstas o que la imagen se recorte. Estas peticiones se procesan en este mismo estado ya que se realiza una secuencia de acciones sin interacción con peticiones intermedias. Cuando la petición es de vídeo, se cambia al estado de Adquisición de imágenes de forma continua. Se vuelve a este estado desde la Adquisición de imágenes cuando se recibe una petición de parada.

Adquisición imágenes : En este estado se realiza de forma continua la adquisición de imágenes para que se realice un vídeo en el SG. Las imágenes se pueden realizar con reconocimiento y/o recorte según las peticiones recibidas desde el SG. De este estado se vuelve al estado de Aceptar peticiones cuando se recibe una petición de parada.

A continuación se detallan las clases más representativas que se han definido en la aplicación.

Manejador de la cámara

Para el acceso a la cámara se ha desarrollado un manejador encapsulado en una clase de C++ denominada CameraSampler. Esta clase se define de la siguiente forma.

Clase:	CameraSampler
Atributos:	int frameRate int videoDevice int dimX, dimY int BrightnessX, Contrast, Saturation int Gain, Exposure
Operaciones:	CameraSampler() bool start() int stop() Mat getImage() bool setBightness(int value) bool setContrast(int value) bool setGain(int value) bool setExposure(int value)

Tabla 4.4: Clase CameraSampler

Lector de peticiones

Para la lectura de las peticiones a ejecutar se ha desarrollado un manejador encapsulado en una clase de C++ denominada CommandListener. Esta clase se define de la siguiente forma.

Clase:	CommandListener
Atributos:	bool newValue byte* value Topic* imgTopic, cmdTopic, resTopic fdeep::model *pModel Camera *pCamera
Operaciones:	CommandListener(Topic* imageTopic, Topic* commandTopic, Topic* responseTopic, fdeep::model *pMod, Camera *pCam, bool verbose) void onMessage(Topic* t, Message* m) Command getCommand() void run() bool newCommand() bool doLoopStep()

Tabla 4.5: Clase CommandListener

4.5 Aplicación de transformación del modelo

Al igual que la aplicación de generación del modelo, la aplicación de transformación del modelo se ha desarrollado de forma independiente para ser ejecutada mediante una orden en línea de comandos. La justificación es la misma que se ha detallado en la de generación del modelo. Posteriormente, como trabajo futuro, se integrará en la aplicación de adquisición y reconocimiento.

La figura 4.21 muestra el esquema de esta aplicación que se ejecuta en el sistema SAR.

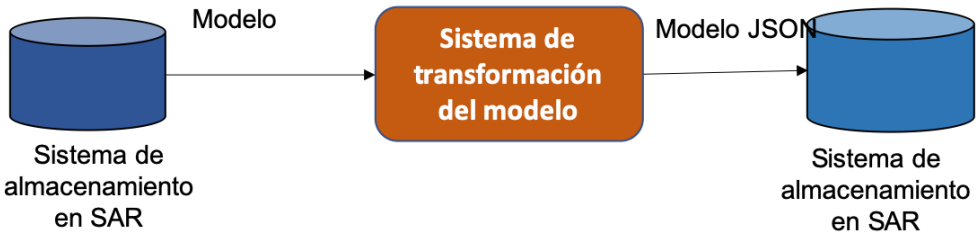


Figura 4.21: Sistema de transformación del modelo

Esta aplicación realiza la transformación del modelo generado mediante la herramienta Keras desde R o Python (formato hdf5) a un formato que se puede integrar en la aplicación en C++ y usarlo para el reconocimiento de imágenes. El procedimiento para ejecutar esta parte del sistema seguirá los siguientes pasos:

1. Se parte de un modelo generado en el Sistema de generación del modelo en formato hdf5 que describe las capas y pesos de la red neuronal. Este fichero está disponible en el SG.

2. Desde una consola del SG, se realiza una copia mediante un conexión remota al SAR (*scp secure copy*) enviándose el fichero del modelo en formato hdf5.

3. En el SAR, copiado el fichero del modelo en formato hdf5, se utiliza una herramienta disponible en la red como es frugally-deep [Her18] que transforma de forma muy rápida un modelo descrito en hdf5 en un modelo JSON en C++ que permite

evitar usar las funciones de Keras al montar la aplicación de adquisición y reconocimiento y usar la librería generada en C++ con las funciones específicas para el reconocimiento de los objetos adquiridos.

4. Con la librería generada, la aplicación de adquisición y reconocimiento se compila y monta con la librería generada en el paso anterior.
5. El resultado del paso anterior es el ejecutable SAR.

Tal y como se ha descrito anteriormente, en el prototipo desarrollado en este TFG, este proceso es totalmente manual. En el trabajo futuro se plantea automatizar el proceso desde el SG y que tanto el sistema de generación como el de transformación del modelo se realicen bajo la acción del usuario pulsando un botón para tal propósito en la interfaz. Al pulsar dicho botón, el SG invocaría las órdenes para

- Añadir un nuevo Topic en el middleware que respondiese al modelo
- Generar el modelo en formato hdf5 en el propio SG
- Publicar el modelo en el middleware
- SAR, como suscriptor del objeto, leería el modelo y aplicaría la herramienta de transformación a JSON.
- SAR recompila la aplicación y la reinicia con el nuevo modelo para el reconocimiento.

Estos serían los pasos en caso de automatizar el proceso, pero, tal y como se ha comentado previamente, esto se plantea como trabajo futuro para el sistema industrial.

4.6 Capa del Middleware

El middleware es un software que proporciona servicios y capacidades comunes a las aplicaciones por encima del sistema operativo. La gestión de datos, los servicios de aplicación, la mensajería y la autenticación son normalmente gestionados por el middleware.

Así pués, el middleware es el componente en el sistema distribuido que se encarga de ofrecer los servicios para la comunicación e interacción entre los distintos subsistemas de una forma transparente y eficiente. El middleware utilizado en este TFG se basa en el trabajo desarrollado en el grupo de Automática e Informática Industrial y se detalla en [Sim18]. A continuación se detallan los aspectos relevantes del diseño e implementación del middleware. Las principales funciones que ofrece el middleware se pueden clasificar en:

- El middleware gestiona información asociada a un tópico *Topic* que permite establecer los mecanismos para la comunicación e interacción. Se pueden definir varios tópicos.
- Los elementos activos (aplicaciones) en cualquier ámbito de direccionamiento pueden conectarse o suscribirse a uno o varios tópicos.
- La publicación de información sobre un tópico por un elemento publicador desencadena el uso de recursos de comunicaciones, que pueden implicar tráfico de red a los suscriptores del tópico independientemente del espacio de direccionamiento en el que se encuentren.

- Los mensajes como resultado de una publicación que reciben los suscriptores vienen afectados por un retardo que depende de la topología de los componentes hardware y software de la aplicación y de la administración de los recursos de comunicaciones que implemente el middleware.

- Un proceso de suscripción implica la asociación de manejadores de eventos que se lanzan cuando se modifica el elemento asociado.

- La ejecución de manejadores de eventos ocasionados por la publicación de información en un nodo remoto debe realizarse forzosamente utilizando un hilo de ejecución suministrado por el middleware.

- Normalmente existe un publicador aunque sería posible definir varios por necesidades de redundancia.

- Un determinado Topic puede tener varios suscriptores. Cada suscriptor se encontrará a una distancia (tiempo de comunicación) diferente de la fuente de la información (publicación) y, por lo tanto, cada uno requerirá del uso de diferentes recursos de comunicaciones para la propagación del evento de actualización. Adicionalmente, cada suscriptor estará interesado en el elemento de información para diferentes fines que requieran diferentes frecuencias de actualización. Por lo tanto, el middleware deberá soportar que cada suscripción especifique en la operación de suscripción un rango de frecuencias admisible de actualización de la información y utilizar los recursos de comunicaciones de forma que no se desperdicie ancho de banda y, además, deberá asegurarse que el publicador de la información del elemento la actualice al menos a la frecuencia más rápida de los suscriptores activos en un determinado momento. Para gestionar esto es necesario que el middleware planifique hilos internos que realicen las tareas de propagación de información.

- En aquellos casos en el que se requieran diferentes restricciones temporales, los mensajes incorporan una prioridad que podrá utilizarse para determinar la clase de tráfico si se usa una infraestructura de red con capacidades de reserva de ancho de banda y para determinar la prioridad de los hilos de ejecución encargados de ejecutar los manejadores del evento.

- Se ofrece un mecanismo de descubrimiento al inicio del middleware consistente en el establecimiento de un canal de difusión *multicast* escogiendo una dirección IP (de tipo "D" desde 224.0.0.0 hasta 239.255.255.255) y un puerto configurando la identidad del grupo. Cada miembro del grupo difunde su identidad enviando su GUID (IP y puerto por el que acepta conexiones como una cadena de caracteres con el formato "XX.XX.XX.XX:PPPP") por este canal de difusión y recibe la identidad de todos los miembros del grupo por este mismo canal. Cuando un par recibe por el canal la identidad de un nuevo par, difunde la suya propia para asegurar que notifica su identidad a los nuevos pares del grupo.

La figura 4.22 muestra el despliegue de aplicaciones en un sistema distribuido.

En la figura se puede ver como las aplicaciones en el mismo o distinto nodo que se han suscrito a un determinado tópico reciben un evento de respuesta cada vez que éste se modifica.

Para una mejor comprensión del diseño e implementación ver [Sim18].

Se han definido 3 *Topic*s que concentran los objetos de la aplicación del middleware. Los tres objetos definidos son:

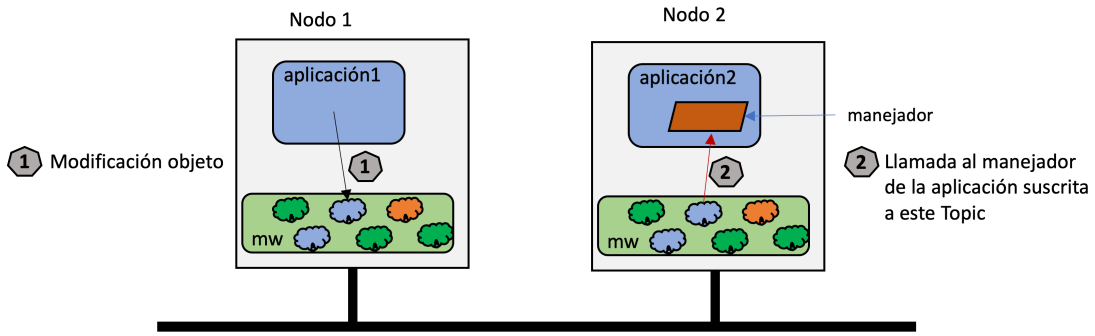


Figura 4.22: Despliegue de aplicaciones sobre un middleware.

COMMAND: Este objeto permite comunicar las órdenes generadas por la aplicación SG y transmitirlas a la aplicación SAR. Las órdenes que pueden ser enviadas a través de este *Topic* son:

- FRAME: Indica que se desea que el SAR adquiera una imagen.
- VIDEO: Indica que se desea que el SAR adquiera una imagen de forma repetitiva para desde el SG crear un video con las imágenes.
- CLASSIFICATION_ON: Indica que se desea que el SAR realice el reconocimiento de la imagen cuando la adquiera.
- CLASSIFICATION_OFF: Indica que se desea que el SAR no realice el reconocimiento de la imagen cuando la adquiera.
- CLIPIMAGE_ON: Indica que se desea que el SAR realice un recorte de la imagen circunscrita al objeto adquirido.
- CLIPIMAGE_OFF: Indica que se desea que el SAR devuelva la imagen tal y como la adquiere en la escena sin un recorte del objeto.
- STOP: Indica que se desea que el SAR finalice la adquisición de imágenes de forma repetitiva.

IMAGE: Este objeto contiene una imagen generada desde el SAR y que es accesible desde el SG. La imagen enviada a través del objeto podrá estar recortada o no según el estado del SAR después de haber recibido una orden para que se realice el recorte o no.

RESPONSE: Este objeto contiene la respuesta del reconocimiento realizado por el SAR a una imagen adquirida. Este objeto se escribe desde el SAR si la última orden recibida para que se realice la clasificación de las imágenes habilita la clasificación de los objetos. La respuesta es un vector de tamaño el número de piezas a reconocer con las probabilidades de que la pieza reconocida sea de cada uno de las clases entrenadas.

En la figura 4.23 se muestra gráficamente los objetos definidos en el middleware y el uso que hacen las aplicaciones de estos objetos. La aplicación SG es un publicador del objeto COMMAND mientras que la aplicación SAR es un suscriptor de este objeto. Cuando un objeto se modifica en uno de los nodos, su nuevo valor se propaga a los nodos en los que hay suscriptores de este objeto. Un hilo de ejecución del middleware en el sistema empotrado se encarga de propagar los cambios del *Topic* COMMAND a la aplicación.

De igual manera, los *Topics* IMAGE y RESPONSE tienen un publicador (en este caso el SAR) y un suscriptor (el SG) con el hilo de ejecución asociado a cada objeto que informa

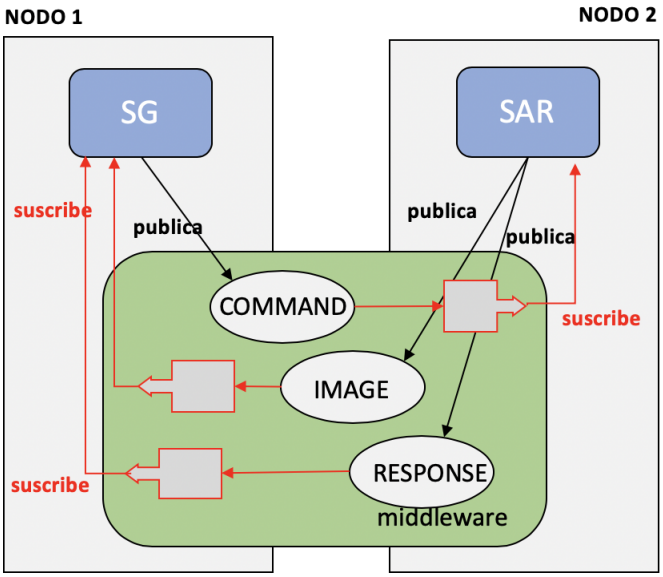


Figura 4.23: *Topics* del middleware

a la aplicación (SG) sobre cambios que se han producido en el NODO_2 y que se han transmitido al NODO_1.

El middleware se instala en cada uno de los nodos que conforman el sistema final. En función del entorno de ejecución de cada nodo, se utilizará la misma versión del middleware implementada con el lenguaje apropiado. En este caso, el nodo de control será una plataforma de propósito general que puede estar implementada con cualquier sistema operativo (Windows, Linux, MacOS) y, por lo tanto, tanto la aplicación como el middleware se implementarán en Java. En los nodos de adquisición y reconocimiento de imágenes, el entorno de ejecución es específico y programado en C++ sobre Linux. Por lo tanto, el middleware usará la versión en C++.

A continuación se detallan las principales funciones de los componentes internos del middleware que permiten la comunicación entre nodos y la propagación de la información.

Paquete Ethernet

El paquete Ethernet agrupa las clases del middleware que están fuertemente ligadas a la comunicación. Las principales clases que la forman son:

Multipeer: Maneja un conjunto de conexiones TCPIP para permitir la difusión de "Mensajes.ª un grupo de destinos.

Destination: Representa un canal bidireccional TCPIP a través del cual se comunican dos entidades. Es necesaria la operación de conectar para que sea efectiva la comunicación. La clase *Multipeer* dispone de una lista de destinos. Cada objeto destino Cada tiene asociado un hilo de ejecución que comunica la recepción de un mensaje mediante el método *OnMesssage*.

IPPort: Encapsula la identidad del par: dirección IP y puerto.

Message: Implementa las propiedades comunes del mensaje.

QueueMessage: Implementa la cola de mensajes.

Paquete DCPS

El paquete denominado DCPS (*Data Centric Publisher Subscripter*) implementa las clases que permiten estructurar los objetos de la aplicación en el middleware. Las clases que lo componen son:

Topic: Permite definir los objetos que se van a ser utilizados en un esquema de un publicador y muchos suscriptores. Cada mensaje publicado en un *Topic* se pone a disposición de cada suscriptor registrado en dicho *Topic*. Los mensajes se envían a un *Topic* y se transmiten a cada uno de los suscriptores. Está identificada por un nombre y soporta los métodos publish() y addListener().

TopicScope: Este objeto en un *MultiPeerListener* que recibe los mensajes del tipo *TOPIC-DATA* y *TOPICLINKS* de un objeto *MultiPeer*. El objeto *TopicScope* mantiene un conjunto de objetos *Topic* y encamina los mensajes correspondientes a los suscriptores que se hayan definido para cada *Topic*. Para crear u obtener una referencia a un objeto *Topic*, se utiliza el método *lookup* de *TopicScope* especificando el nombre del *Topic*. Si el *Topic* ya existe en el *TopicScope*, devolverá una referencia al objeto existente, si no existe, crea uno y devuelve una referencia al objeto creado.

TopicListener: Es una clase abstracta que fuerza a implementar el método *onMessage(Topic t, Message m)* para asociar un método cuando se reciba el mensaje.

4.7 Conclusiones

En este capítulo se ha planteado el diseño de todas las piezas que forman el proyecto para proceder a su implantación.

En este capítulo se ha planteado el diseño de los distintos subsistemas y componentes que conforman el sistema completo. El diseño ha tenido como objetivo el definir las interfaces de los distintos componentes y detallar, en aquellos que lo requerían, la especificación de su comportamiento. Inicialmente se ha presentado la arquitectura global y los principales componentes que la conforman. Para cada componente se ha presentado una vista de los elementos internos y sus interfaces. En aquellos los componentes principales se ha definido mediante un diagrama de estados su especificación que es la base para su implementación. también se ha detallado la interfaz de usuario con sus elementos gráficos.

Respecto al módulo de generación del modelo, se han detallado los fundamentos de la técnica usada y los procesos internos que lo forman. Por último se ha descrito las principales funcionalidades e interfaces de la capa de middleware utilizada en el sistema final.

CAPÍTULO 5
Implantación

En este apartado se detalla la implementación del sistema final. En primer lugar se detalla la vista de despliegue en el que detallan los sistemas involucrados. A continuación se detalla la implementación de la parte de generación del modelo en las dos versiones en las que se ha realizado en los lenguajes de programación R y Python. Tanto las aplicaciones de control como de adquisición y reconocimiento se aportarán en el anexo con el código desarrollado. En este apartado solo se ofrecerá una visión general de la implementación.

5.1 Vista de despliegue del sistema global

La vista de despliegue del sistema muestra la disposición física de los componentes. La figura 5.1 muestra la relación de nodos que pueden intervenir en la vista física del sistema.

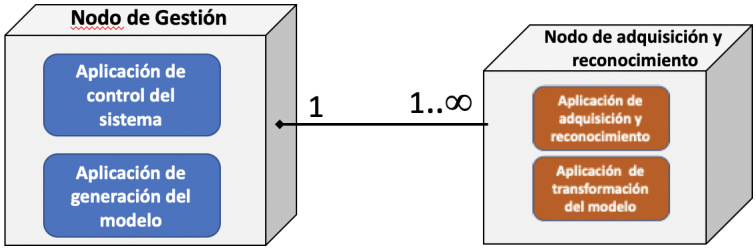


Figura 5.1: Diagrama físico del sistema

El nodo de gestión identifica un computador con un sistema operativo de propósito general ya que las aplicaciones se han desarrollado en el lenguaje Java que es multiplataforma. Este nodo se conecta a un número indeterminado de nodos específicos que realizan las funciones de adquisición y reconocimiento de imágenes. Estos nodos son sistemas empotrados integrados por una RaspberryPi 4 con las siguientes características.

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz,
- 1GB, 2GB or 4GB LPDDR4-3200 SDRAM
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE Gigabit Ethernet
- v2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header

- 2 micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- Micro-SD card slot for loading operating system and data storage
- Módulo de Cámara para Raspberry Pi 5MP 1080p OV5647 Sensor Cámara de Vídeo de HD Soporte Visión Nocturna

5.2 Módulo de generación del modelo en lenguaje R

A partir de un conjunto de imágenes adquiridas y organizadas en un árbol de directorios en el que todas las figuras de una clase determinada se disponen en un directorio con el nombre de la clase, se realiza el proceso de generación del modelo descrito en el apartado 4.3. El flujograma detallado en 4.7 describe el algoritmo de generación del modelo.

Inicializa las variables asociadas a la ubicación en el sistema de archivos de las imágenes.

```
dir_training <- "train"
dir_validation <- "validation"
dir_testing <- "test"

chuches_list <- dir(dir_training)
numclases <- length(chuches_list)
```

Genera imágenes a partir de una de ella con distintas posiciones, giros y tamaños.

```
  dataGen_training <- image_data_generator(
  rotation_range = 40,
  width_shift_range = 0.2,
  height_shift_range = 0.2,
  shear_range = 0.2,
  zoom_range = 0.2,
  horizontal_flip = TRUE,
  fill_mode = "nearest"
)
```

Especifica el conjunto de imágenes que se asocian al conjunto de entrenamiento, prueba y validación.

```
generator4training <- flow_images_from_directory(dir_training,
    generator = dataGen_training, target_size = c(img_width, img_height),
    color_mode = "rgb", class_mode = "categorical",
    batch_size = batch_size, shuffle = TRUE, seed = 123)

generator4testing <- flow_images_from_directory(dir_testing,
    generator = image_data_generator(), target_size = c(img_width, img_height),
    color_mode = "rgb", classes = NULL, class_mode = "categorical",
    batch_size = batch_size, shuffle = TRUE, seed = 123)

generator4validation <- flow_images_from_directory(dir_validation,
    generator = image_data_generator(), target_size = c(img_width, img_height),
    color_mode = "rgb", classes = NULL, class_mode = "categorical",
    batch_size = batch_size, shuffle = TRUE, seed = 123)
```

Define el modelo a utilizar para la generación del modelo. Este modelo es el secuencial definido en la herramientas Keras.

```
model <- keras_model_sequential()
```

Define el número de componentes del modelo identificado con tres capas de tamaño 16, 32 y 64, respectivamente.

```
model %>%
  layer_conv_2d(filter = 16, kernel_size = c(3,3), input_shape = c(img_width,
      img_height, 3)) %>%
  layer_activation("relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  layer_conv_2d(filter = 32, kernel_size = c(3,3)) %>%
  layer_activation("relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  layer_conv_2d(filter = 64, kernel_size = c(3,3)) %>%
  layer_activation("relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  layer_flatten() %>%
  layer_dense(64) %>%
  layer_activation("relu") %>%
  layer_dropout(0.5) %>%
  layer_dense(numclases) %>%
  layer_activation("softmax")
```

Se compila el modelo.

```
model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_adam(),
  metrics = "accuracy"
)
```

Se define un histograma para representar la evolución del proceso de generación del modelo y se conecta con el proceso de reconocimiento.

```
histogram <- model %>% fit_generator(
  generator4training,
  steps_per_epoch = 64,
  epochs = 50,
  validation_data = generator4testing,
  validation_steps = 48,
  verbose = 2
)

histogram <- data.frame(acc = unlist(histogram$metrics$acc),
    val_acc=unlist(histogram$metrics$val_acc),
    val_loss = unlist(histogram$metrics$val_loss),
    loss = unlist(histogram$metrics$loss))

colnames(histogram) <- c("Training Accuracy", "Validation Accuracy",
    "Validation Loss", "Training Loss")

histogram$Epoch <- 1:nrow(histogram)

m <- gather(histogram, variable, value, -Epoch)
```

Se visualizan los resultados mediante los histogramas definidos.

```
ggplot(m, aes(Epoch, value)) +
```

```
    facet_wrap(~variable, as.table = T, scales = "free" ) +
    geom_point() + geom_smooth() + ylab("") + theme_light()
```

El modelo generado se guarda en el disco en formato hdf5.

```
save_model_hdf5(model,"Model")
```

Se presentan los resultados finales del modelo obtenido.

```
ggplot(result,aes(x=confianza,group=original)) +
  geom_density(aes(color=original))

ggplot(result,aes(x=original,y=clase))+
  geom_jitter(aes(color=confianza))+
  scale_color_viridis()
```

Se realiza el proceso de evaluación del modelo generado.

```
validation_samples <- 15
evaluate_generator(model,generator4testing, validation_samples)
```

Las dependencias son las detalladas a continuación.

```
library(tidyverse)
library(keras)
library(viridisLite)
library(viridis)
```

En el capítulo 6 se detallan los resultados de la aplicación del modelo al conjunto de imágenes.

5.3 Módulo de generación del modelo en lenguaje Python

En esta sección se detalla el mismo algoritmo de generación del modelo descrito en el apartado 4.3 en el lenguaje Python. Una de las acciones es comparar las prestaciones de ambas soluciones.

Define los directorios de trabajo y carga las imágenes.

```

dirname = os.path.join(os.getcwd(), 'train')
imgpath = dirname + os.sep
(epochs, batch_size) = getOpts(sys.argv)

images = []
directories = []
dircount = []
prevRoot=''
cant=0

for root, dirnames, filenames in os.walk(imgpath):
    for filename in filenames:
        if re.search("\.(jpg|jpeg|png|bmp|tiff)$", filename):
            cant=cant+1
            filepath = os.path.join(root, filename)
            image = plt.imread(filepath)
            images.append(image)
            if prevRoot !=root:
                print(root, cant)
                prevRoot=root
                directories.append(root)
```

```
                dircount.append(cant)
                cant=0

dircount.append(cant)

dircount = dircount[1:]
dircount[0]=dircount[0]+1
```

Crea las etiquetas de las clases de objetos.

```
labels=[]
indice=0
for cantidad in dircount:
    for i in range(cantidad):
        labels.append(indice)
    indice=indice+1
```

Crea el corpus de las imágenes a generar el modelo.

```
chuches=[]
indice=0
for directorio in directories:
    name = directorio.split(os.sep)
    chuches.append(name[len(name)-1])
    indice=indice+1

y = np.array(labels)
X = np.array(images, dtype=np.uint8)

# reduce las clases repetidas a una  nica  opci n
classes = np.unique(y)
nClasses = len(classes)
```

Crea los conjuntos de entrenamiento y de test y los preprocesa para ajustar los al rango 0..1

```
train_X, test_X, train_Y, test_Y = train_test_split(X,y,test_size=0.2)

train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255.
test_X = test_X / 255.
```

Cambia las etiquetas de categorical a one-hot encoding ajustando los conjuntos anteriores.

```
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

train_X, valid_X, train_label, valid_label = train_test_split(train_X,
    train_Y_one_hot, test_size=0.2, random_state=13)
```

Se configuran los parámetros del modelo y niveles.

```
# nivel de frecuencia de aprendizaje
INIT_LR = 1e-3
# Cantidad de iteraciones completas al conjunto de imagenes de entrenamiento
epochs = 10
# cantidad de im genes que se toman a la vez en memoria
batch_size = 64

# modelo y capas
chuche_model = Sequential()
```

```
chuche_model.add(Conv2D(16, kernel_size=(3, 3),activation='linear',padding='
    same',input_shape=(64,64,3)))
chuche_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',padding='
    same',input_shape=(64,64,3)))
chuche_model.add(Conv2D(64, kernel_size=(3, 3),activation='linear',padding='
    same',input_shape=(64,64,3)))

chuche_model.add(LeakyReLU(alpha=0.1))
chuche_model.add(MaxPooling2D((2, 2),padding='same'))
chuche_model.add(Dropout(0.5))

chuche_model.add(Flatten())
chuche_model.add(Dense(64, activation='linear'))
chuche_model.add(LeakyReLU(alpha=0.1))
chuche_model.add(Dropout(0.5))
chuche_model.add(Dense(nClasses, activation='softmax'))

chuche_model.summary()
chuche_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=
    keras.optimizers.Adagrad(lr=INIT_LR, decay=INIT_LR / 100),metrics=['
    accuracy'])
```

Se procesa el modelo y se guarda en el fichero de salida con formato hdf5

```
chuche_train = chuche_model.fit(train_X, train_label, batch_size=batch_size,
    epochs=epochs,verbose=1,validation_data=(valid_X, valid_label))

chuche_model.save("modelo_chuche.h5py")
```

Se convierte el modelo en formato JSON.

```
model_json = chuche_model.to_json()
with open("modelo_chuche.json", "w") as json_file:
    json_file.write(model_json)
```

Se realiza la evaluación del modelo y se visualiza mediante histogramas.

```
test_eval = chuche_model.evaluate(test_X, test_Y_one_hot, verbose=1)

accuracy = chuche_train.history['accuracy']
val_accuracy = chuche_train.history['val_accuracy']
loss = chuche_train.history['loss']
val_loss = chuche_train.history['val_loss']
epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Las dependencias de otros módulos de esta implementación son las detalladas a continuación.

```
import numpy as np
import os
import re
import getopt
import sys
```

```
import ast

import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

import keras
from keras.utils import to_categorical
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
```

5.4 Fragmentos de código de los listeners

Los manejadores de eventos (*listeners*) permiten ejecutar un código ante modificaciones de los objetos a los que se está suscrito. A continuación se detallan las partes más relevantes de éstos en la aplicación del sistema de adquisición y reconocimiento escrito en el lenguaje de programación C++.

```
enum Command {
        STOP = 0,
        VIDEO = 1,
        FRAME = 2,
        CLASSIFICATION_ON = 3,
        CLASSIFICATION_OFF = 4,
        CLIPIMAGE_ON = 5,
        CLIPIMAGE_OFF = 6
};

// define la clase CommandListener con sus operaciones
class CommandListener: public TopicListener {
public:
  CommandListener(Topic* imageTopic, Topic* commandTopic, Topic* responseTopic,
       fdeep::model *pMod, Camera *pCam, bool verbose);
   virtual ~CommandListener();

// cuando se recibe un mensaje se ejecuta onMessage
  void onMessage(Topic* t, Message* m);
// devuelve la orden recibida
  Command getCommand();
// ejecuta la orden
  void run();
// procesa una imagen
  void processFrame();
//mira si hay nuevas  rdenes
  bool newCommand();
//ejecuta la adquisi n de forma repetida para que se
// genere una secuencia o video
  bool doLoopStep();

//Atributos de la clase
private:
  Command cmd;
  bool newValue;
  byte* value;
  bool continousVideo;
  //
    std::mutex m;
    std::condition_variable cond_var;
```

```
  //
  Topic* imgTopic;
  Topic* cmdTopic;
  Topic* resTopic;
  fdeep::model *pModel;
  Camera *pCamera;
  bool verbose;
  bool doClassifiction;
  bool doClipImgROI;
  std::string lastOLEDLabel;

};
```

De todas estas operaciones se detalla a continuación la de *onMessage* que pressenta mayor visibilidad de lo que realiza.

```

void CommandListener::onMessage(Topic* t, Message* m) {
  byte* value;

    lock();

  value = m->getBufferPayload();

  if (value[0] == '1') {
    //VIDEO
    continousVideo = true;
    cond_var.notify_all();
    cmd = VIDEO;
  }
  else if (value[0] == '2') {
    //FRAME
    this->doLoopStep();
    cmd = FRAME;
  }
  else if (value[0] == '3') {
    //STOP
    continousVideo = false;
    cmd = STOP;
  }
  else if (value[0] == '4') {
    //CLASSIFICATION_ON
    doClassifiction = true;
    cmd = CLASSIFICATION_ON;
  }
  else if (value[0] == '5') {
    //CLASSIFICATION_OFF
    doClassifiction = false;
    cmd = CLASSIFICATION_OFF;
  }
  else if (value[0] == '6') {
    //CLIPIMAGE_ON
    doClipImgROI = true;
    cmd = CLIPIMAGE_ON;
  }
  else if (value[0] == '7') {
    //CLIPIMAGE_OFF
    doClipImgROI = false;
    cmd = CLIPIMAGE_OFF;
  }
  newValue = true;
    unlock();
}
```

Este código sigue el modelo definido en la máquina de estados definida en la figura 4.20.

5.5 Adquisición de la imagen en el sistema empotrado

La clase CameraSampler define la clase encargada de manejar la adquisión de imágenes en el sistema empotrado. A continuación se detallan las operaciones y algunos atributos relevantes. Otros como el brillo, luminosidad, etc., no se detallan en estas líneas.

```
class CameraSampler : public Camera {
public:
  CameraSampler();
  virtual ~CameraSampler();
  bool start();
  int stop();
  Mat getImage();

  bool setBightness(int value);
  .....

  double getApparentFPS() {return apparentFPS;}
  friend void *cameraSamplingThread(void * arg);
private:
  Mat buff1, buff2, buff3;
  //otros atributos no detallados
```

5.6 Reconocimiento de la imagen

EL proceso de reconocimiento de la imagen adquirida se realiza en el subsistema de adquisición y reconocimiento descrito en la sección 4.4. Uno de los componentes fundamentales es este subsistema es el reconocimiento de la imagen adquirida. Este reconocimiento se realiza mediante la ejecución de la red neuronal generada en el proceso de entrenamiento fuera de linea. Tal y como se ha descrito en la sección 4.5 la red neuronal generado en el SG es enviada al SAR para que el modelo sea transformado de un formato en formato hdf5 en el que se detallan las capas y los pesos de los nodos de la capas en un formato que se puede integrar en la aplicación en C++ y usarlo para el reconocimiento de imágenes.

En el SAR, a partir del modelo en formato hdf5, se realiza la conversión del modelo mediante la herramienta frugally-deep [Her18] qenera un modelo descrito en JSON en C++. El resultado es una librera de funciones específicas que evita que el proceso de reconocimiento no requiera usar las funcionalidades de Keras y que con un código más eficiente integra en el proceso de generación del ejecutable los servicios para el reconocimiento de las imágenes usando los servicios de la librería generada.

Los servicios de reconocimiento se han integrado en un componente denominado *ClassificationFunctions* que se detalla a continuación.

La interfaz del componente se incluye en el listado siguiente.

```
#include <opencv2/opencv.hpp>
#include <fdeep/fdeep.hpp>

using namespace std;
using namespace cv;
```

```
6
7  // Funciones::
8
9  // Devuelve un string con el resultado del proceso de inferencia
10 std::string inferenceResultToString(fdeep::tensor5, bool onlyLabel = false);
11
12 // Convierte un objeto Mat de OpenCV en un buffer de bytes para ser
13 // procesados por frugallyDeep
14 std::uint8_t *matToFdeepBuffer(Mat &image, int dimx, int dimy);
15
16 // Carga desde un fichero un buffer de bytes para ser
17 // procesado por frugallyDeep
18 std::uint8_t *loadFdeepBuffer(const char * filename, int * read);
19
20 // Muestra la imagen descrita por el buffer de bytes
21 void showFdeepBuffer(std::uint8_t * buff, int dimx, int dimy, string wname, int
       twait);
22
23 // Realiza el proceso de clasificaci n de la imagen almacenado
24 // en un objeto Mat de OpenCV
25 fdeep::tensor5 classifyImage(fdeep::model mod, Mat &image, int dimx, int dimy,
      bool verbose);
26
27 // Realiza la carga del modelo descrito en JSON
28 fdeep::model loadJsonModel(std::string modelFilename);
```

La implementación de las funciones mas relevantes de este componente se detalla en el listado que se detalla a continuación.

```
1  #include "ClassificationFunctions.h"
2  #include <opencv2/opencv.hpp>
3  #include <fdeep/fdeep.hpp>
4
5  using namespace std;
6  using namespace cv;
7
8  std::string inferenceResultToString(fdeep::tensor5 r, bool onlyLabel) {
9  // Resultado de la inferencia
10 // Nombrado de las clases que componen el experimento
11     int nClasses = 14;
12     string classesNames[nClasses];
13     classesNames[0] = "Beso";    classesNames[1] = "Bola";
14     classesNames[2] = "Cola";    classesNames[3] = "Corazon";
15     classesNames[4] = "Dedo";    classesNames[5] = "Dentadura";
16     classesNames[6] = "Fresa";   classesNames[7] = "Melon";
17     classesNames[8] = "Mora";    classesNames[9] = "Nube";
18     classesNames[10] = "Palo";   classesNames[11] = "Platano";
19     classesNames[12] = "Sandia"; classesNames[13] = "Vacio";
20
21     std::vector<std::uint8_t> res = fdeep::tensor5_to_bytes(r);
22     int sz = res.size();
23     if (sz != nClasses) {
24         std::cout << "printInferenceResult: ERROR. Unexpected number of
               classes found. "<< sz << " instead " << nClasses << std::endl;
25     }
26
27     int iMax = -1;
28     double max = 0.0;
29     for (int i = 0; i<sz; i++) {
30         if (r.get(0,0,0,0,i) > max) {
31             max = r.get(0,0,0,0,i);
32             iMax = i;
33         }
34     }
```

```
35
36      std::ostringstream ss;
37      ss << "Clase:\t"+ classesNames[iMax%nClasses] + "\t (";
38      for (int i = 0; i<sz; i++) {
39        char strNum[20];
40        sprintf(strNum,"%1.2f",r.get(0,0,0,0,i));
41        ss <<  strNum;
42          if (i != sz -1) ss <<  ", ";
43      }
44      ss <<  ")";
45      //Devuelve el string con el resultado
46      if (onlyLabel) { return classesNames[iMax%nClasses];}
47      else {return ss.str();}
48  }
49
50  // Convierte un objeto Mat de OpenCV en un buffer de bytes para ser
51  // procesados por frugallyDeep
52  std::uint8_t *matToFdeepBuffer(Mat &image, int dimx, int dimy)
53  {
54      Mat dest(dimx,dimy, CV_8UC3);
55      // redefine el tano si es mayor
56      if ((image.cols != dimx) || (image.rows != dimy) ) {
57        resize(image, dest, Size(dimx,dimy));
58    } else {
59      image.copyTo(dest);
60    }
61      int length = dimx * dimy * 3;
62      std::uint8_t *pChars = new std::uint8_t[length];
63      int count = 0;
64      for (int i=0;i<dest.cols; i++) {
65      for (int j=0; j<dest.rows; j++) {
66                  unsigned char * p = dest.ptr(i, j);
67                  pChars[count] = p[2]; count++;
68                  pChars[count] = p[1]; count++;
69                  pChars[count] = p[0]; count++;
70      }
71    }
72      return pChars;
73  }
74
75  // Carga desde un fichero un buffer de bytes para ser
76  // procesado por frugallyDeep
77  std::uint8_t *loadFdeepBuffer(const char * filename, int * read)
78  {
79      // el codigo se detalla en el anexo
80  }
81
82  // Muestra la imagen descrita por el buffer de bytes
83  void showFdeepBuffer(std::uint8_t * buff, int dimx, int dimy, string wname, int
        twait)
84  {
85      // el codigo se detalla en el anexo
86  }
87
88  // Realiza la carga del modelo descrito en JSON
89  fdeep::model loadJsonModel(std::string modelFilename) {
90    return fdeep::load_model(modelFilename);
91  }
92
93  // Realiza el proceso de clasificaci n de la imagen almacenado
94  // en un objeto Mat de OpenCV
95  fdeep::tensor5 classifyImage(fdeep::model mod, Mat &image, int dimx, int dimy,
      bool verbose) {
96      std::uint8_t *buff;
```

```
97      // Convertir la imagen a raw buffer
98      buff = matToFdeepBuffer(image,dimx,dimy);
99      // Visualiza el buffer generado
100     if (verbose) showFdeepBuffer(buff, dimx, dimy, "Buffer del jpg",1);
101     fdeep::tensor5 myTensor = fdeep::tensor5_from_bytes(buff, dimx, dimy, 3, 0,
            255);
102     const auto result = mod.predict({myTensor});
103     delete[] buff;
104     return(result[0]);
105 }
```

5.7 Prototipo realizado: fotografías del montaje

Las siguientes figuras muestra los distintos trabajos y montajes realizados con el fin de poder organizar, probar y validar el sistema prototipado.

La figura 5.3 muestra el sistema de iluminación diseñado para el escenario de adquisición de imágenes,

Figura 5.2: Sistema de iluminación del prototipo

La figura 5.3 muestra el sistema con objeto a reconocer.

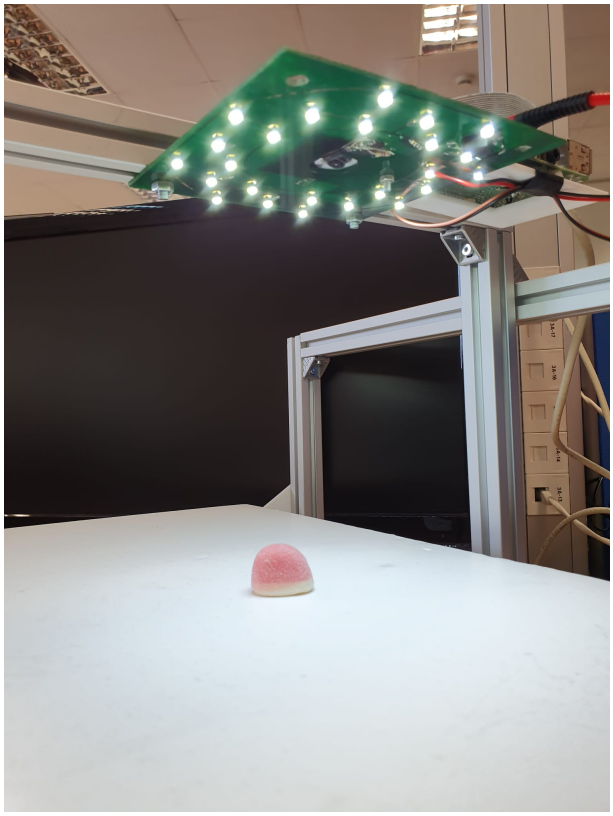


Figura 5.3: Prototipo preparado para la adquisición de un objeto.

La figura 5.4 muestra la vista cenital del SAR con la Raspberry y la cámara para la adquisición de imágenes.

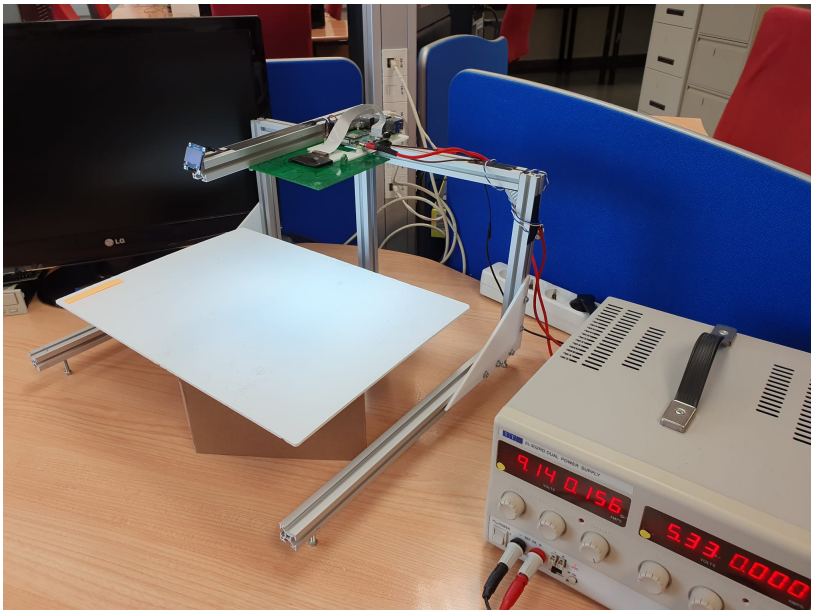


Figura 5.4: El Sistema de adquisición y reconocimiento

La figura 5.5 muestra la vista cenital del SAR con la Raspberry y la cámara para la adquisición de imágenes.

Figura 5.5: Imagen del prototipo

5.8 Conclusiones

A partir de la vista de despliegue del sistema, en este capítulo se han presentado las partes principales de la implementación del prototipo realizado. Se ha obviado el detalle del código desarrollado que se presenta en el Anexo. En cambio, se ha destacado la implementación realizada, en ambos lenguajes R y Python, de la generación del modelo por resultar mas novedosa y requerir un detalle explicativo que se ha acompañado al detallar la implementación.

Asimismo, se han presentado los fragmentos de código de los manejadores de eventos que el middleware genera a los suscriptores de mensajes y la adquisición de imágenes. Por último se ha presentado mediante distintas fotografías el prototipo físico realizado junto con el escenario montado para la adquisición de imágenes.

CAPÍTULO 6
Pruebas

En este apartado se presentarán las pruebas que se hayan realizado para verificar que la solución funciona correctamente, y las pruebas de validación (con el usuario) si se han realizado para comprobar que el sistema realiza lo que el usuario espera y lo que se ha especificado. También se pueden presentar pruebas de carga para comprobar su eficiencia y el consumo de recursos.

6.1 Conjunto de datos

El conjunto de datos *Dataset* es una colección de objetos que se utilizan para entrenar, validar o testear el modelo de RNN generado. En este caso es una colección de imágenes formadas por distintos tipos de golosinas. La colección completa está formada por 15 clases de golosinas, cada una con un identificador diferente que hace referencia a la golosina que representa. Cada clase está formada por 200 imágenes hechas en diferentes posiciones y con diferentes intensidades de luz.

En la tabla 6.1 podemos ver varios ejemplos de cada clase.

Como uno de los objetivos del TFG es evaluar distintas alternativas para el reconocimiento de objetos, se compararán los resultados entre las dos versiones del algoritmo de entrenamiento implementadas en R y Python.

Las pruebas a realizar tienen las siguientes características:

- Número de iteraciones: Se realizarán un número de iteraciones que será función del número de clases. El mínimo de iteraciones será de 10.
- Número de imágenes de cada clase: 200.
- Método de cálculo de las perdidas en el modelo: el método usado es el *categorical_crossentropy* que se calcula con una función de activación Softmax y un método de entropia cruzada. La siguiente expresión define este método:

  $CE = -log(\frac{e^{(S_p)}}{\Sigma_j^C (e^{(S_j)}}$

- Método de cálculo de la precisión en el modelo: el método usado para la determinación de la precisión es el denominado *accuracy* que permite calcular la frecuencia de predicción de las etiquetas. Esta función de precisión crea dos variables locales, total y contador, que se utilizan para calcular la frecuencia con la que las predicciones coinciden con las etiquetas. Esta frecuencia se devuelve finalmente como exactitud: una operación idempotente que simplemente divide el total por el contador.

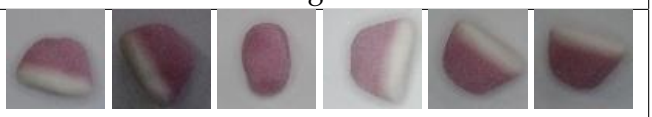
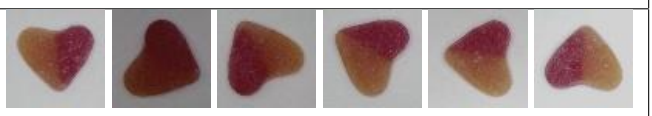
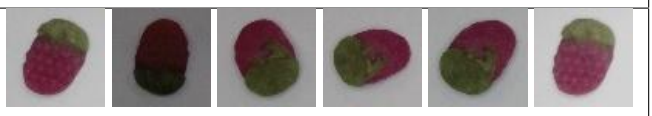
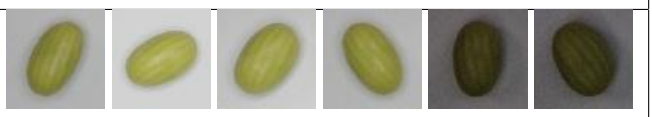
Identificador	Clase	Imágenes
00	Beso	
01	Cola	
02	Corazón	
03	Dedo	
04	Dentadura	
05	Mora	
06	Palo	
07	Vacío	
08	Fresa	
09	Moneda	
10	Nube	
11	Sandía	
12	Bola	
13	Plátano	
14	Melón	

Tabla 6.1: Dataset del entrenamiento.

Para la generación del modelo de red neuronal se va a entrenar la red con distintos números de clases de piezas: 2, 4, 8 , 12, 13 y 15. Lo que se pretende obtener es la relación entre número de clases, número de iteraciones y el resultado del proceso de entrenamiento en cuanto a porcentajes y el coste temporal.

Siguiendo la filosofía de este proyecto que tenía como objetivo el montaje de un prototipo que sirviese de base para una implantación industrial futura, el conjunto de imágenes se han obtenido mediante el propio prototipo desarrollado. Este prototipo ha sido usado para la adquisición de las imágenes que han permitido el entrenamiento para la generación de los modelos de red neuronal en situaciones similares a las de un funcionamiento en un entorno industrial. Además el propio sistema es usado para la fase de reconocimiento de imágenes.

6.2 Módulo de generación del modelo en lenguaje R

En esta sección se mostrarán los resultados del proceso de entrenamiento utilizando el desarrollo realizado en el lenguaje de programación R. Se han realizado unos experimentos con distintos números de clase a entrenar con el fin de analizar y comparar los resultados obtenidos.

6.2.1. Entrenamiento con 2 clases

En esta sección se exponen los resultados obtenidos utilizando 2 clases para el entrenamiento.

En la figura 6.1 se representan dos gráficas que representan la evolución con el número de iteraciones (épocas) de las funciones de pérdida del conjunto de entrenamiento (*loss*) y del conjunto de validación (*val_loss*) y de la función de precisión o acierto del conjunto de entrenamiento (*acc*) y del de validación (*val_acc*). Como se puede ver con pocas iteraciones los valores de pérdida y la precisión alcanzan unos valores excelentes.

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
2	10	200	64

Tabla 6.2: Ficha del experimento para 2 clases.

Al realizar el proceso con pocas clases el número de iteraciones será del mínimo establecido en el marco de pruebas.

Los resultados finales de este experimento se representan en la tabla 6.3.

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
100 %	100 %	0.10 %	0,00002	94 s

Tabla 6.3: Resultados del entrenamiento con 2 clases

6.2.2. Entrenamiento con 4 clases

En esta sección se exponen los resultados obtenidos utilizando 4 clases para el entrenamiento.

Al igual que en el caso de 2 clases, se considera que un número de iteraciones de 10 es suficiente para alcanzar los niveles de precisión apropiadas.

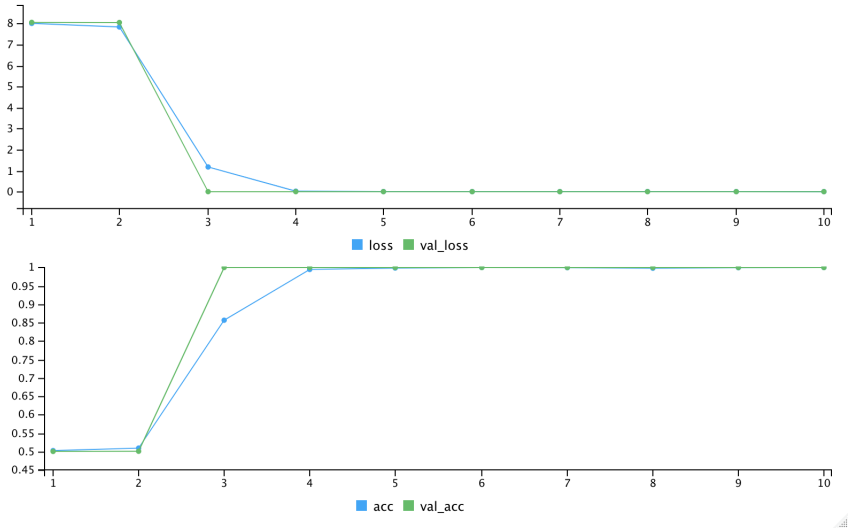


Figura 6.1: Entrenamiento con 2 clases

Las gráficas muestran la evolución con el número de iteraciones (épocas) de las funciones de pérdida del conjunto de entrenamiento (*loss*) y del conjunto de validación (*val_loss*) y de la función de precisión o acierto del conjunto de entrenamiento (*acc*) y del de validación (*val_acc*).

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
4	10	200	64

Tabla 6.4: Ficha del experimento para 2 clases.

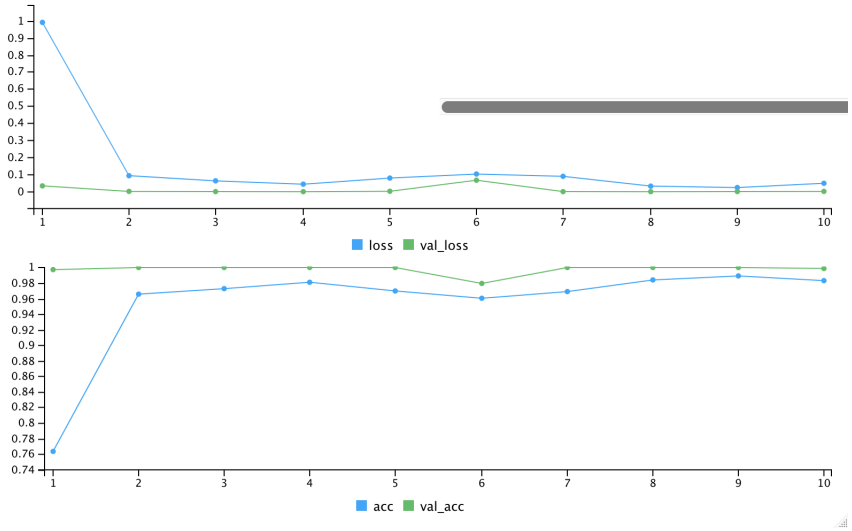


Figura 6.2: Entrenamiento con 4 clases

Los resultados obtenidos se resumen en la tabla 6.5.

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
98.32 %	99.87 %	4.95 %	0.15 %	110 s

Tabla 6.5: Resultados del entrenamiento con 4 clases

6.2.3. Entrenamiento con 8 clases

En esta sección se exponen los resultados obtenidos utilizando 8 de las clases de los objetos para el entrenamiento.

En este caso, al haber aumentado el número de clases del experimento el número de iteraciones es de 15.

Las gráficas (6.3) muestran la evolución con el número de iteraciones de las funciones de pérdida del conjunto de entrenamiento (*loss*) y del conjunto de validación (*val_loss*) y de la función de precisión o acierto del conjunto de entrenamiento (*acc*) y del de validación (*val_acc*).

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
8	15	200	64

Tabla 6.6: Ficha del experimento para 8 clases.

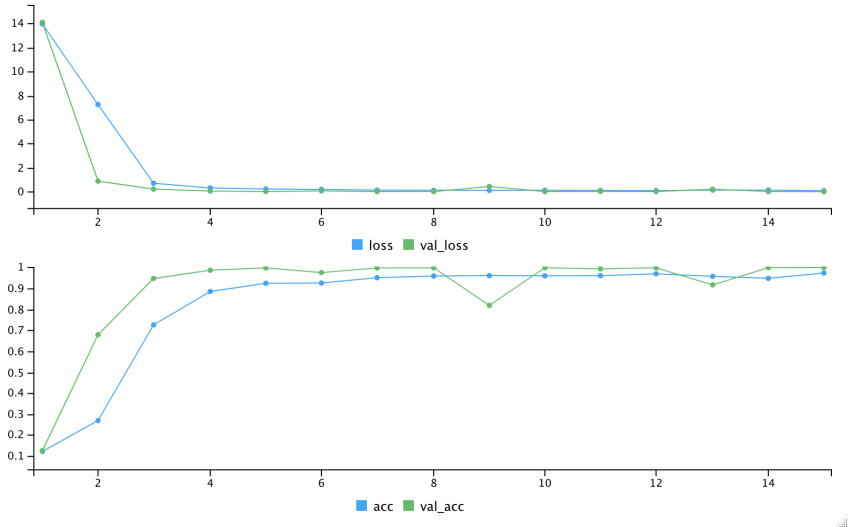


Figura 6.3: Entrenamiento con 8 clases

Al igual que en los casos anteriores, la tabla 6.7 resume los resultados obtenidos.

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
97.36 %	99,14 %	7.19 %	0.17 %	139 s

Tabla 6.7: Resultados del entrenamiento con 8 clases

6.2.4. Entrenamiento con 10 clases

En esta sección se exponen los resultados obtenidos utilizando 10 de las clases de los objetos para el entrenamiento. En este caso, el número de iteraciones es de 20.

Las gráficas (6.4) muestran la evolución con el número de iteraciones de las funciones de pérdida del conjunto de entrenamiento (*loss*) y del conjunto de validación (*val_loss*) y de la función de precisión o acierto del conjunto de entrenamiento (*acc*) y del de validación (*val_acc*).

Los características del entrenamiento se datallan en 6.4.

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
10	20	200	64

Tabla 6.8: Ficha del experimento para 10 clases.

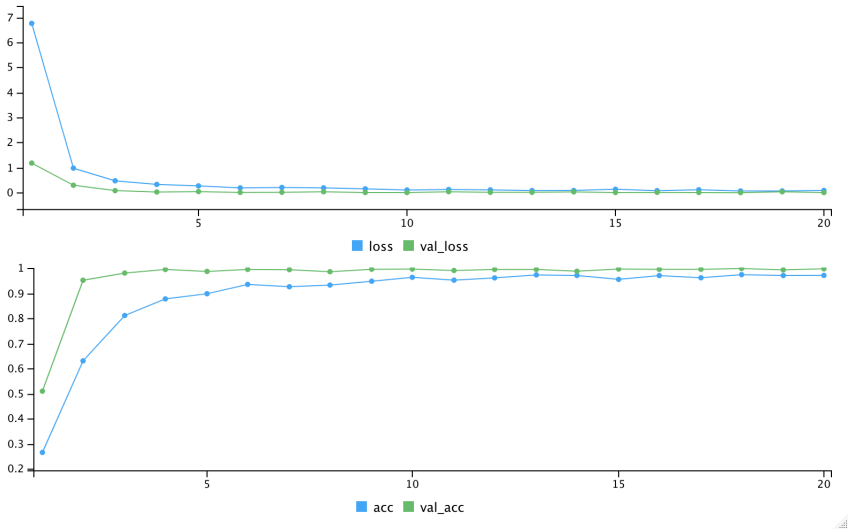


Figura 6.4: Entrenamiento con 10 clases

La tabla 6.9 resume los resultados obtenidos para 10 clases.

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
97.31 %	98.82 %	8.79 %	0.15 %	146 s

Tabla 6.9: Resultados del entrenamiento con 10 clases

6.2.5. Entrenamiento con 12 clases

En esta sección se exponen los resultados obtenidos utilizando 12 de las clases de los objetos para el entrenamiento. En este caso, el número de iteraciones es de 40.

Las gráficas (6.5) muestran la evolución con el número de iteraciones de las funciones de pérdida del conjunto de entrenamiento (*loss*) y del conjunto de validación (*val_loss*) y de la función de precisión o acierto del conjunto de entrenamiento (*acc*) y del de validación (*val_acc*).

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
12	40	200	64

Tabla 6.10: Ficha del experimento para 12 clases.

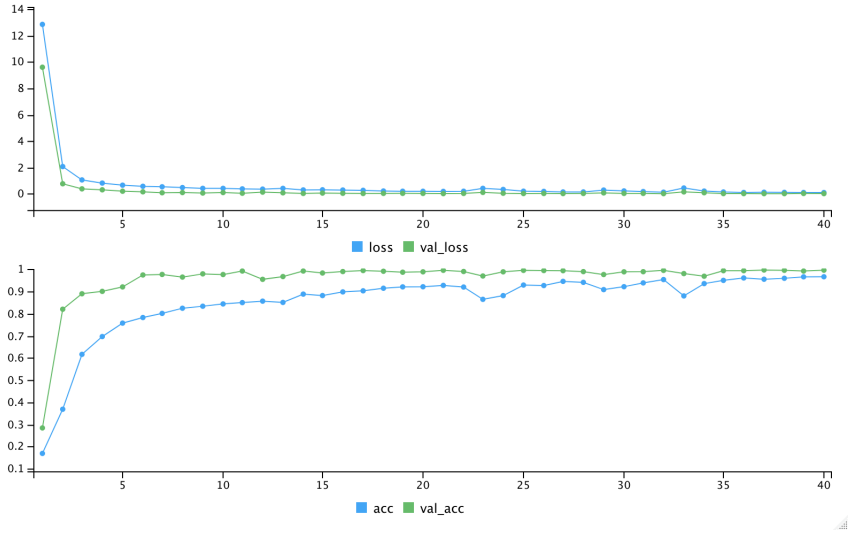


Figura 6.5: Entrenamiento con 12 clases

La tabla 6.11 resume los resultados obtenidos para 12 clases.

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
96.73 %	98.18 %	9.40 %	1.21 %	375 s

Tabla 6.11: Resultados del entrenamiento con 12 clases

6.2.6. Entrenamiento con 15 clases

En esta sección se presentan los resultados obtenidos utilizando las 15 clases que conforman el conjunto de datos *DataSet* del problema definido en la tabla 6.1.

Las gráficas (6.6) muestran la evolución con el número de iteraciones de las funciones de pérdida del conjunto de entrenamiento (*loss*) y del conjunto de validación (*val_loss*) y de la función de precisión o acierto del conjunto de entrenamiento (*acc*) y del de validación (*val_acc*).

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
15	40	200	64

Tabla 6.12: Ficha del experimento para 15 clases.

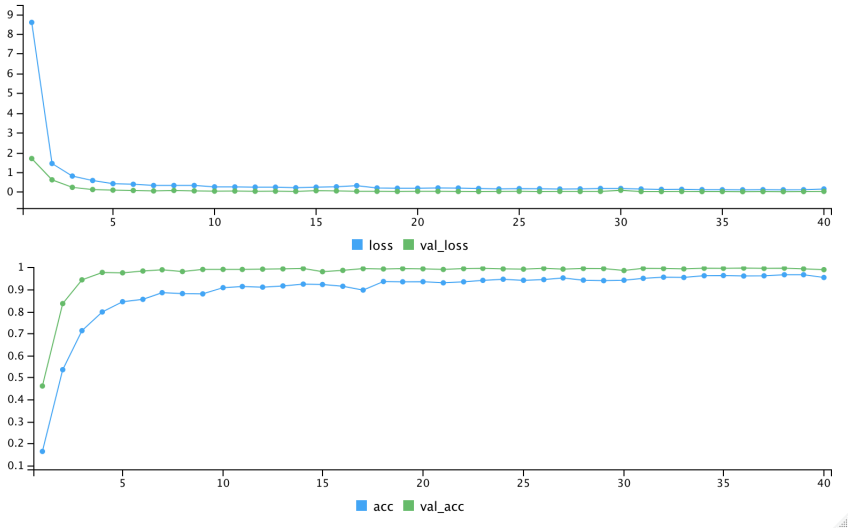


Figura 6.6: Entrenamiento con el DataSet del problema

La tabla 6.13 resume los resultados obtenidos para 15 clases.

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
95.46 %	98.02 %	14.48 %	2.11 %	467 s

Tabla 6.13: Resultados del entrenamiento con 15 clases

6.3 Módulo de generación del modelo en lenguaje Python

Al igual en la sección anterior, se presentan los resultados obtenidos utilizando el programa realizado en Python. El objetivo es comparar en resultados de precisión, pérdida y tiempo de cómputos ambas soluciones. Al final se ofrecerá una tabla comparativa de resultados.

La diferencia de la representación gráfica es debida a la herramienta gráfica disponible en cada uno de los entornos. Mientras que el entorno del programa en R se visualiza la evolución en el tiempo, en la versión en Python se representa el estado final al no poder representar la evolución con el número de iteraciones.

6.3.1. Entrenamiento con 2 clases

Resultados obtenidos utilizando las 2 clases con 10 iteraciones.

Las gráficas (6.7) muestran la evolución con el número de iteraciones de las funciones de pérdida del conjunto de entrenamiento (*loss*) y del conjunto de validación (*val_loss*) y de la función de precisión o acierto del conjunto de entrenamiento (*acc*) y del de validación (*val_acc*).

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
2	10	200	64

Tabla 6.14: Ficha del experimento para 2 clases.

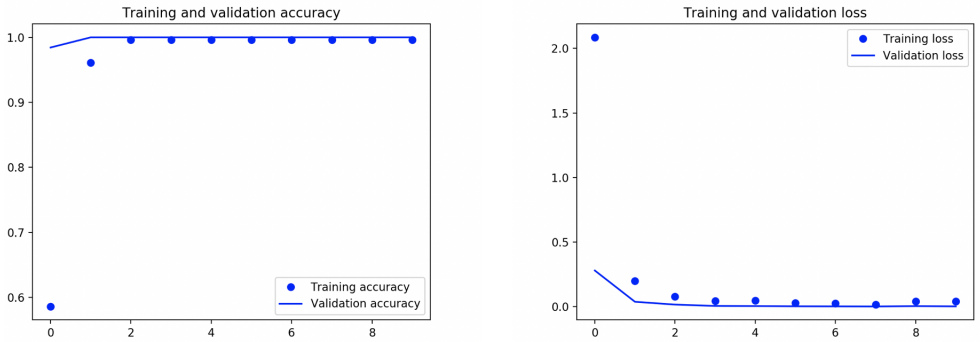


Figura 6.7: Entrenamiento con 2 clases

La tabla 6.15 resume los resultados obtenidos para 2 clases.

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
99.61 %	100 %	2.05 %	0.025 %	22 s

Tabla 6.15: Resultados del entrenamiento con 2 clases

6.3.2. Entrenamiento con 4 clases

Las gráficas (6.8) muestran los resultados obtenidos utilizando las 4 clases con 10 iteraciones.

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
4	10	200	64

Tabla 6.16: Ficha del experimento para 4 clases.

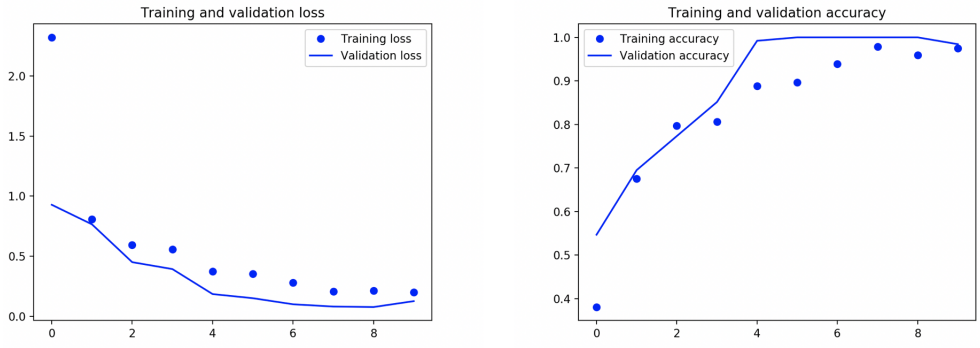


Figura 6.8: Entrenamiento con 4 clases

La tabla 6.17 resume los resultados obtenidos para 4 clases.

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
97.46 %	98.44 %	5.21 %	1.71 %	41 s

Tabla 6.17: Resultados del entrenamiento con 4 clases

6.3.3. Entrenamiento con 8 clases

Las gráficas (6.9) muestran los resultados obtenidos utilizando las 8 clases con 15 iteraciones.

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
8	15	200	64

Tabla 6.18: Ficha del experimento para 2 clases.

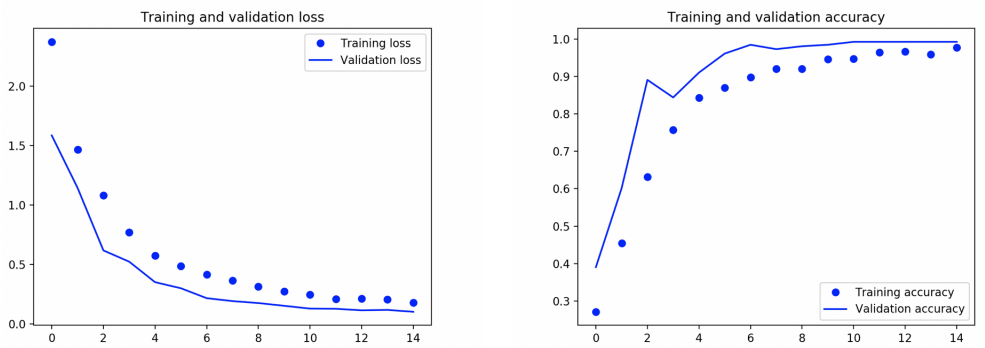


Figura 6.9: Entrenamiento con 8 clases

La tabla 6.19 resume los resultados obtenidos para 8 clases.

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
97.28 %	98.12 %	8.22 %	1.23 %	62 s

Tabla 6.19: Resultados del entrenamiento con 8 clases

6.3.4. Entrenamiento con 10 clases

Las gráficas (6.10) muestran los resultados obtenidos utilizando las 10 clases con 20 iteraciones.

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
10	20	200	64

Tabla 6.20: Ficha del experimento para 10 clases.

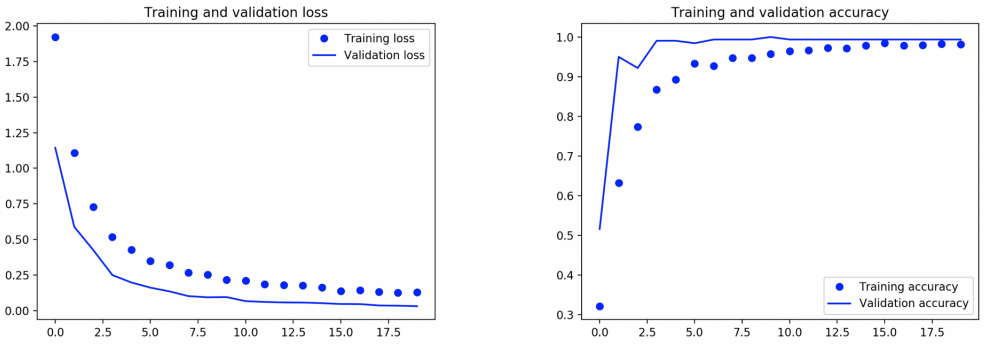


Figura 6.10: Entrenamiento con 10 clases

La tabla 6.21 resume los resultados obtenidos para 10 clases.

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
96.21 %	97.89 %	9.24 %	1.56 %	71 s

Tabla 6.21: Resultados del entrenamiento con 10 clases

6.3.5. Entrenamiento con 12 clases

Las gráficas (6.11) muestran los resultados obtenidos utilizando las 12 clases con 40 iteraciones.

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
12	40	200	64

Tabla 6.22: Ficha del experimento para 12 clases.

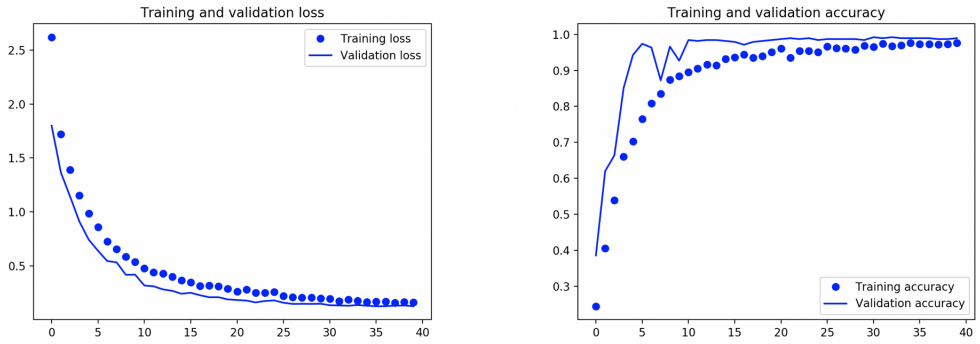


Figura 6.11: Entrenamiento con 12 clases

La tabla 6.23 resume los resultados obtenidos para 12 clases.

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
96.03 %	97.70 %	10.38 %	1.89 %	84 s

Tabla 6.23: Resultados del entrenamiento con 12 clases

6.3.6. Entrenamiento con 15 clases

Las gráficas (6.12) muestran los resultados obtenidos utilizando las 15 clases del problema y con 40 iteraciones.

Clases	Iteraciones	Imágenes por clase	Cantidad de imágenes procesadas a la vez
15	40	200	64

Tabla 6.24: Ficha del experimento para 15 clases.

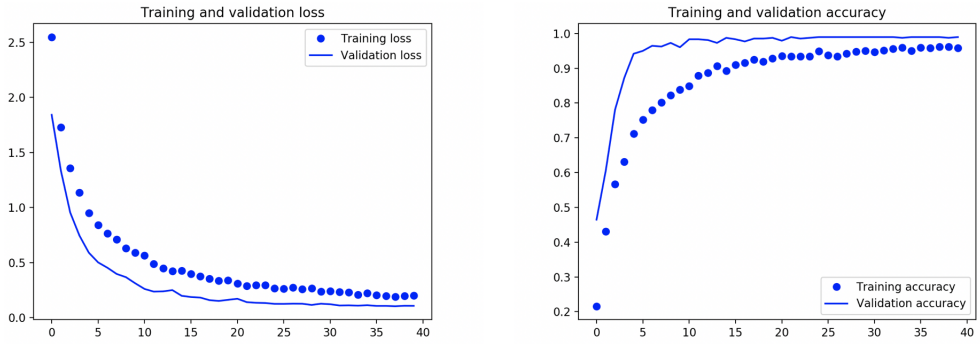


Figura 6.12: Entrenamiento con 15 clases

La tabla 6.25 resume los resultados obtenidos para todas las clases del problema..

Training accuracy	Validation accuracy	Training loss	Validation loss	Time (seconds)
96.01 %	97.12 %	12.21 %	2.19 %	101 s

Tabla 6.25: Resultados del entrenamiento con 15 clases

6.4 Resumen de los resultados

A continuación se comparan los resultados obtenidos con los desarrollos en los lenguajes de programación R y Python. La tabla 6.26 resume esta información.

Lang.	Clases	Train. accuracy	Val. accuracy	Train. loss	Val. loss	Time (s)
R2	2	100 %	100 %	0.10 %	0,00002	94 s
R4	4	98.32 %	99.87 %	4.95 %	0.15 %	110 s
R8	8	97.36 %	99,14 %	7.19 %	0.17 %	139 s
R10	10	97.31 %	98.82 %	8.79 %	0.15 %	146 s
R12	12	96.73 %	98.18 %	9.40 %	1.21 %	375 s
R15	15	95.46 %	98.02 %	14.48 %	2.11 %	467 s
P2	2	99.61 %	100 %	2.05 %	0.025 %	22 s
P4	4	97.46 %	98.44 %	5.21 %	1.71 %	41 s
P8	8	97.28 %	98.12 %	8.22 %	1.23 %	62 s
P10	10	96.21 %	97.89 %	9.24 %	1.56 %	71 s
P12	12	96.03 %	97.70 %	10.38 %	1.89 %	84 s
P15	15	96.01 %	97.12 %	12.21 %	2.19 %	101 s

Tabla 6.26: Síntesis de los resultados

A continuación se presentan y comentan los resultados obtenidos al generar el modelo en ambos módulos desarrollados.

La figura 6.13 compara la precisión obtenida en función del número de clases entrenadas y discriminando los resultados por versión en el conjunto de entrenamiento.

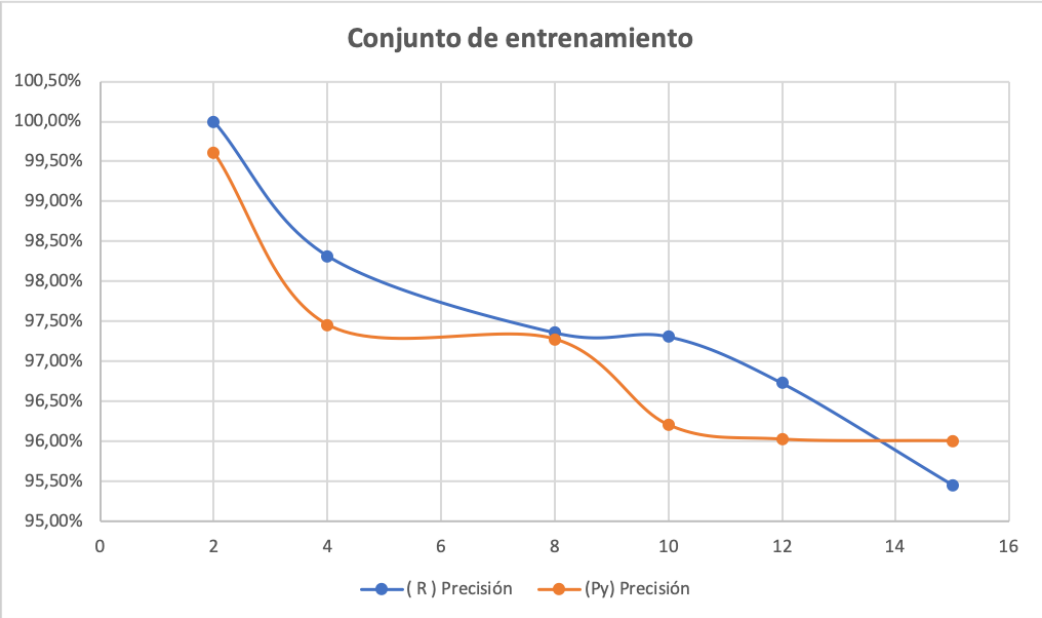


Figura 6.13: Precisión en el conjunto de entrenamiento

La respuesta muestra un comportamiento prácticamente igual de la precisión con el número de clases. La máxima diferencia se obtiene en el número de 4 y 10 clases en el que la diferencia es menor del 1.5 % lo que es algo de poca relevancia.

La figura 6.14 compara la perdida obtenida en función del número de clases entrenadas y según la versión en el conjunto de entrenamiento.

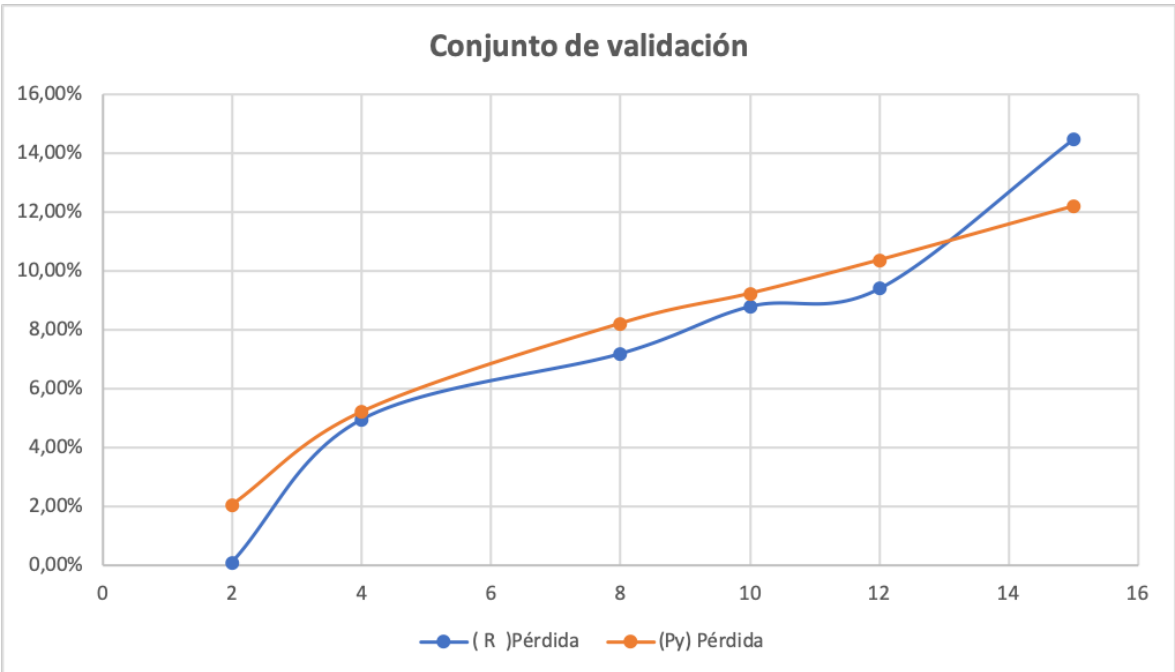


Figura 6.14: Pérdida en el conjunto de entrenamiento

El mismo comportamiento en las dos soluciones se observa con las pérdidas del proceso de entrenamiento. La máxima diferencia se obtiene con todas las clases con un valor aproximado del 2 %.

La figura 6.15 compara la precisión obtenida en función del número de clases entrenadas y discriminando los resultados por versión en el conjunto de validación.

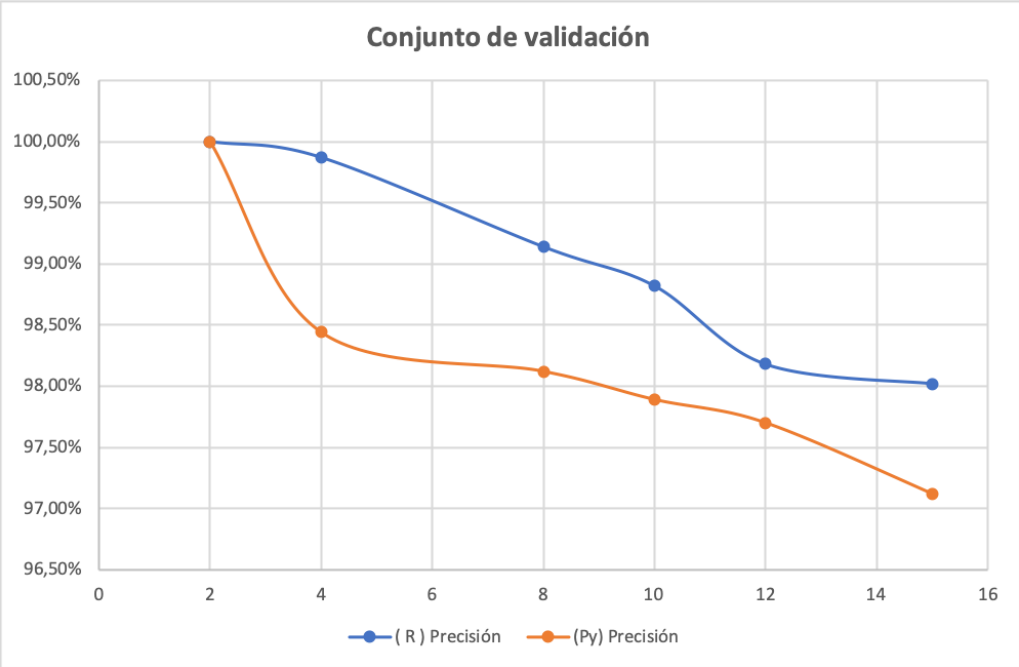


Figura 6.15: Precisión en el conjunto de validación

Los resultados mostrados ofrecen la precisión cuando se aplica sobre el conjunto de validación. Se observa un mejor comportamiento en todos los casos de la versión en R. Para todas las clases, la diferencia es del orden de 2 %.

La figura 6.16 compara la perdida obtenida en función del número de clases entrenadas y según la versión en el conjunto de validación.

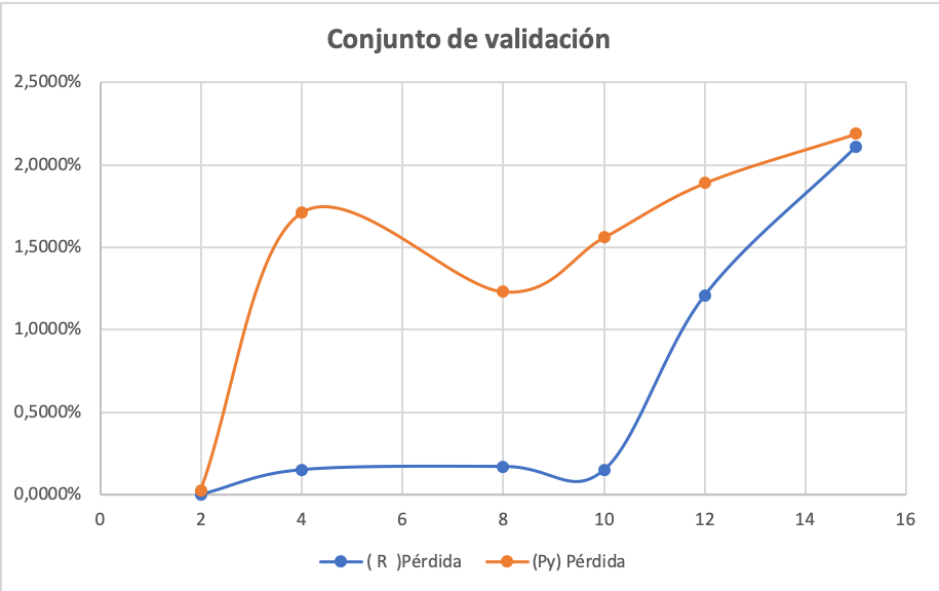


Figura 6.16: Pérdida en el conjunto de entrenamiento

Esta figura muestra las pérdidas dependiendo del número de clases. Se observa grandes diferencias con pocas clases que disminuye a medida que el número de clases aumenta. Para el conjunto total, la diferencia es mínima entre las dos soluciones.

Por último, la figura 6.17 compara los tiempos de obtención del modelo en función del número de clases entrenadas en el conjunto de entrenamiento.

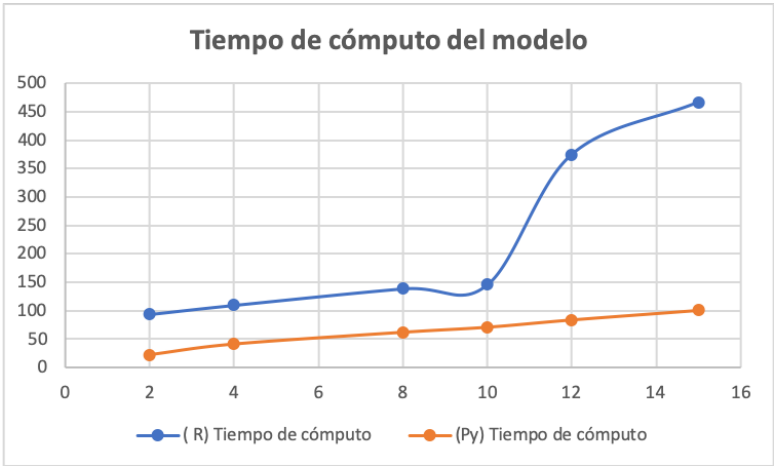


Figura 6.17: Tiempo de cómputo en el cálculo del modelo

Los resultados muestran en general un mejor comportamiento de la versión en Python respecto a R sobre todo a partir de un número de clases significativo (mayor de 10). La gráfica 6.18 muestra el porcentaje de incremento/decremento respecto a la versión de 15 clases que representa el conjunto total de clases que se ha tomado como referencia.

Figura 6.18: Tiempo de cómputo en porcentaje

Como puede apreciarse en la gráfica y tomando el 100 % el valor de la versión de Python para las 15 clases, se muestra el incremento importante del coste la versión en R.

Esto puede tener una explicación debido a que en el entorno de R se realiza un trazado de los valores a cada iteración mientras que la versión en Python no se realiza y se genera la información al final de todas las iteraciones.

En cualquier caso, cabe recordar la relativa importancia de los tiempos de cómputo al tratarse de un proceso que se produce fuera de línea previo al proceso de reconocimiento que se realiza a partir del modelo generado en este proceso.

6.5 Evaluación del sistema final

En la sección anterior se ha evaluado el proceso de generación del modelo que se concreta en la implementación de la red neuronal que es transformada en C++ para integrada, compilada y ejecutada por el sistema SAR por la Raspberry Pi.

En esta sección se analizan los tiempos de ejecución del sistema cuando está en operación, esto es, cuando se adquiere la imagen, se hace el reconocimiento, y se devuelve la información al sistema de gestión (SG) para su tratamiento posterior.

En primer lugar se realizará un análisis de las actividades involucradas en una operación desde que el operario en la interfaz solicita una imagen hasta que la interfaz la guarda en disco o la presenta en la propia interfaz. Las actividades y, por lo tanto, los tiempos de cada una de ellas son:

Tiempo de carga del modelo : t_{mod}. Es el tiempo de carga del modelo de la red neuronal desde el fichero que describe en JSON. Este tiempo no es solamente indicativo ya que la carga del modelo se realiza una vez en la inicialización del SAR y, por lo tanto, no afecta a la respuesta de peticiones desde el SG.

Tiempo de publicación de la orden : t_{pub}^{Orden}. Es el tiempo invertido en la trasmisión de publicación del mensaje de 1 byte con la publicación de la orden en el middleware.

Tiempo de trasmisión de la orden : t_{trn}^{Orden}. Es el tiempo necesario para el envío a través de la interfaz de red del mensaje de 1 byte con la petición de la orden publicada en el middleware.

Tiempo de adquisición de imagen : t_{adq}^{Imagen}. Es el necesario en el SAR para realizar la adquisición de la imagen utilizando la cámara previamente inicializada en la fase anterior de configuración.

Tiempo de reconocimiento de la imagen : t_{rec}^{Imagen}. Es el tiempo requerido para una vez adquirida la imagen se ejecuta el código generado a partir del modelo generado.

Tiempo de recorte (clip) de la imagen : t_{clp}^{Imagen}. Es el tiempo requerido para tratar la imagen mediante un algoritmo de recorte de la imagen.

Tiempo de publicación de la imagen : t_{pub}^{Imagen}. Es el tiempo requerido para publicar la imagen en el middleware del SAR.

Tiempo de transmisión de la imagen : t_{trn}^{Imagen}. Es el tiempo necesario para el envío a través de la interfaz de red del mensaje con la imagen. El tamaño de la imagen tendrá un tamaño que depende de la resolución de la cámara (1280 x 960 bytes) o un tamaño limitado (64 x 64 bytes) en case de haber aplicado un recorte de la imagen.

Tiempo de lectura de la imagen : t_{lct}^{Imagen}. Es el tiempo necesario para que el manejador lea la imagen del middleware y la copie al buffer de la aplicación del SG.

Tiempo de mostrado de la imagen en la interfaz : t_{vis}^{Imagen}. Es el tiempo necesario para que el SG muestre la imagen en la interfaz.

Evidentemente, muchos de estos tiempos no son aplicables ya que dependen de las opciones seleccionadas por el usuario en la interfaz. Algunas de estas acciones, principalmente las de publicación y comunicación van acompañadas de una marca de tiempo (*timestamp*) que el middleware inserta automáticamente. Por lo tanto, estos tiempos ya están disponibles en el propio mensaje. Los otros se requiere instrumentar el código para poder determinarlos.

Como el objetivo es el análisis del problema de tiempo real, es necesario determinar el tiempo de la peor situación de las operaciones o tiempo de peor caso que incluye la selección de todas las opciones que introducen tiempo. Por lo tanto, el tiempo de peor caso será:

$$t_{TOTAL} = t_{pub}^{Orden} + t_{trn}^{Orden} + t_{adq}^{Imagen} + t_{rec}^{Imagen} + t_{clp}^{Imagen} + t_{pub}^{Imagen} + t_{trn}^{Imagen} + t_{lct}^{Imagen} + t_{vis}^{Imagen} \tag{6.1}$$

Los tiempos de publicación, suscripción y trasmisión de peor caso serán cuando la imagen se envíe de forma completa sin recortar. La opción de recorte se realiza siempre e implica que lo que se envía como respuesta es la imagen recortada.

Mientras que todos los tiempos deberían ser casi predecibles, el tiempo de trasmisión a través de una red con protocolo Ethernet puede estar sometido a una cierta aleatoriedad. Sin embargo en un entorno como el del prototipo con una única interfaz de red, la trasmisión debería comportarse de forma similar. En caso de su industrialización este aspecto debería analizarse en detalle.

6.5.1. Tiempo de carga del modelo

Al iniciar la aplicación del SAR, el siguiente listado (6.5.1) muestra los mensajes generados en el que se detalla el tiempo de carga y de construcción de las estructuras del modelo.

```
$ ./SAR -m Model.json
OLED1306::initI2C: Bus Connected to SSD1306
Camera connected.
Loading json ... done. elapsed time: 0.093316 s
Running test 1 of 1 ... done. elapsed time: 0.018037 s
Loading, constructing, testing of Model.json took 0.178930 s overall.
```

En el listado se puede identificar el tiempo de carga del modelo desde el disco y el tiempo de construcción del modelo. En este caso, los tiempos son 93316 y 178930 $\mu segundos$ respectivamente. El tiempo total de carga $t_{mod} = 272246 \mu segundos$.

La tabla 6.27 se muestra los resultados de 10 ejecuciones de SAR en el se muestran los tiempos y se calculan el promedio, máximo y mínimo. Los resultados se dan en microsegundos y para un modelo del número máximo de 15 clases.

No. Prueba	T. Carga	T. Cons.	T. Carga modelo
1	91139	161432	252571
2	97300	166400	263700
3	82809	144630	227439
4	125525	192778	318303
5	90328	159632	249960
6	93316	178930	272246
7	93659	172640	266299
8	90916	162403	253319
9	90860	160580	251440
10	130793	195406	326199
Promedio	**98664**	**169483**	**268147**
Máximo	**130793**	**195406**	**326199**
Mínimo	**82809**	**144630**	**227439**

Tabla 6.27: Síntesis de los resultados

Por lo tanto, aunque el tiempo es para la fase de inicialización, la carga y construcción del modelo se puede estimar en el peor caso observado es menor de 0.4 segundos.

6.5.2. Tiempos de ejecución en SG

La evaluación de los distintos tiempos involucrados en la operativa de petición desde el SG de una imagen y la obtención de la imagen tal y como se ha descrito en la sección 6.5.

Para la realización de los experimentos se han definido cuatro perfiles o casos que representan las distintas opciones del usuario en la interfaz. Los casos definidos son los siguientes:

- Caso 1: Muestra la imagen que hace la cámara.
- Caso 2: Muestra la imagen recortado a 64x64 px.
- : Caso 3: Muestra la imagen tal y como la hace la cámara y nos muestra el resultado del reconocimiento.
- : Caso 4: Muestra la imagen recortado a 64x64 px y nos muestra el resultado del reconocimiento.

Las pruebas se han realizado con 3 de los objetos (coca-cola, dentadura y beso) con el modelo generado para las 15 clases. Se han realizado 10 experimentos para cada uno de los objetos en cada uno de los 4 casos.

El Caso 3 representa el caso considerado como peor desde el punto de vista del tiempo de cómputo ya que se realizan todas las operaciones y se envía la imagen completa.

La tabla 6.28 muestra los resultados del Caso 3 para la clase dentadura. El tiempo que se muestra corresponde a los microsegundos que tarda el proceso desde que se pide una imagen y su reconocimiento hasta que se muestra por pantalla junto con el resultado.

No. Prueba	Tiempo t_{TOTAL}
1	1448887
2	1283047
3	1489997
4	1346733
5	1489997
6	1386989
7	1178124
8	1154334
9	1077969
10	1242410
Promedio	**1309848**
Máximo	**1489997**
Mínimo	**1077969**

Tabla 6.28: Caso 3. Dentadura. (Valores en microsegundos)

Esta misma prueba se ha realizado para los otros 2 objetos de referencia y los resultados se resumen en la tabla 6.29.

	Clase	Tiempo
Promedio	Dentadura	1309848
Promedio	Cola	1267787
Promedio	Beso	1284709
Máximo	Dentadura	1489997
Máximo	Cola	1547792
Máximo	Beso	1539159
Mínimo	Dentadura	1077969
Mínimo	Cola	1120765
Mínimo	Beso	1090403

Tabla 6.29: Caso 3. Comparativa. (Valores en microsegundos)

Como podemos observar en la tabla, el sistema es muy estable en cuanto a tiempo se refiere. El tipo de pieza no es excesivamente relevante en el tiempo que tarda el proceso. Los tiempos promedio se encuentran alrededor de un segundo con algunas décimas. El peor caso observado en los experimentos es menor de 1.6 segundos. El mejor tiempo observado de poco mas de 1 segundo.

Por otra parte el tiempo de visualización de la imagen t_{vis}^{Imagen} se ha medido de forma independiente para la imagen de tamaño mayor (correspondiente a la dentadura). Los valores obtenidos para 25 ejecuciones se resumen en un promedio de 3.3, un máximo de 3.8 y un mínimo de 2.9 μs.

6.5.3. Tiempos de ejecución SAR

Para la toma de muestra de los tiempos de las distintas fases del procesamiento se ha instrumentado el código para poder adquirir el tiempo antes y después de la operación y se ha calculado la diferencia. El listado 6.5.3 muestra el esquema seguido para la toma de los tiempos (adquisición de imagen en este ejemplo). Se ha usado un atributo adicional (tAcq en este caso) en la clase para guardar la diferencia de cada operación y poder mostrarlos al final del proceso por la consola.

```
$ ./SAR -m Model.json
  auto ta1 = chrono::steady_clock::now();
  Mat image = pCamera->getImage();
  auto ta2 = chrono::steady_clock::now();
  tAcq = chrono::duration_cast<chrono::nanoseconds>(ta2 - ta1).count();
```

Los tiempos que se han medido son los correspondientes al tiempo de adquisición de la imagen (t_{adq}^{Imagen}), el tiempo de procesamiento para el recorte de la imagen t_{clp}^{Imagen} correspondiente a la operación *processBox*, el tiempo de clasificación o reconocimiento t_{rec}^{Imagen}, el tiempo de publicación de la imagen t_{pub}^{Imagen} y el tiempo total t_{TOTAL}^{SAR} desde que llega la petición de la imagen y esta se resuelve después de la publicación. En este momento es cuando se imprimen los resultados. Esta instrumentación del código para la medida de los tiempos de las distintas operaciones introduce un tiempo adicional que se evaluará en la siguiente sección.

El tiempo total obtienen como la suma de los tiempos descritos en la siguiente expresión.

$$t_{TOTAL}^{SAR} = t_{adq}^{Imagen} + t_{rec}^{Imagen} + t_{clp}^{Imagen} + t_{pub}^{Imagen} \tag{6.2}$$

El listado 6.5.3 muestra la salida por consola de los resultados obtenidos una vez se ha instrumentado el código utilizando el esquema descrito en 6.5.3. Estos resultados se han capturado y guardado para su posterior tratamiento.

```
$ ./SAR -m Model.json
T-Total : 246642225 ns; 246642 us; 246 ms
T-Clip : 118082621 ns; 118082 us; 118 ms;
T-Rec : 17877237 ns; 17877 us; 17 ms;
T-Pub : 534084 ns; 534 us; 0 ms;
```

El tiempo de adquisición de la cámara t_{adq}^{Imagen} es independiente de las distintas opciones de la orden recibida desde SG. El tamaño de la imagen adquirida es el mismo en todos los casos. Por ello, se han realizado un conjunto de pruebas para medir este tiempo. Se han realizado 25 experimentos con distintos imágenes y los resultados obtenidos han permitido calcular los resultados promedio: 7.1 μs, máximo: 11 μs y mínimo: 6 μs.

Para el resto de tiempos se han procesado los distintos casos descritos anteriormente. La tabla 6.30 muestra los resultados del Caso 3 para la clase dentadura. Se ha elegido este caso ya que, como se ha comentado previamente, corresponde al peor caso de todos.

No. Prueba	t_{TOTAL}^{SAR}	t_{clp}^{Imagen}	t_{rec}^{Imagen}	t_{pub}^{Imagen}
1	378281	100378	15142	152063
2	859145	106859	15449	517643
3	984683	148035	19849	692255
4	366861	106078	15539	134614
5	397864	130402	15942	140958
6	862711	139084	20463	592953
7	367786	109042	15582	133708
8	798204	102239	14356	571339
9	369300	103189	15142	140051
10	910274	110726	14495	674707
Promedio	**629510**	**115603**	**16195**	**375029**
Máximo	**984683**	**148035**	**20463**	**692255**
Mínimo	**366861**	**100378**	**14356**	**133708**

Tabla 6.30: Caso 3. Dentadura. (Valores en microsegundos)

Se ha elegido el Caso 3 ya que se miden todos los elementos, incluido el recorte de la imagen, pero se envía la imagen completa y no la recortada como corresponde al caso 4. El Caso 3 es el peor escenario que requiere más tiempo de proceso.

Si se comparan los Caso 3 de los 3 objetos medidos se obtiene la tabla 6.31

	Clase	t_{TOTAL}^{SAR}	t_{clp}^{Imagen}	t_{rec}^{Imagen}	t_{pub}^{Imagen}
Promedio	Dentadura	629510	115603	16195	375029
Promedio	Coca	517881	195748	16972	193032
Promedio	Beso	547059	151803	16342	268669
Máximo	Dentadura	984683	148035	20463	692255
Máximo	Coca	674516	226995	20096	374460
Máximo	Beso	849155	174109	20564	564120
Mínimo	Dentadura	366861	100378	14356	133708
Mínimo	Coca	464485	166302	14516	138192
Mínimo	Beso	407835	128595	14461	135081

Tabla 6.31: Caso 3. Comparativa. (Valores en microsegundos)

De la comparativa se puede extraer que los tiempos de reconocimiento de la imagen (t_{rec}^{Imagen}) son muy estables independientemente de la clase de objeto. Los tiempos de recorte (t_{clp}^{Imagen}) dependen del objeto y su posición en la escena aunque no varían demasiado. En cambio los tiempos de publicación (t_{pub}^{Imagen}) tienen una gran variabilidad

A continuación se comparan los promedios, valores máximos y mínimos de los casos 1,2 y 4.

La tabla6.32 muestra los valores resumen del Caso 1. En este caso no se aportan los valores de las opciones no seleccionas (recorte y reconocimiento),

	Clase	t_{TOTAL}^{SAR}	t_{clp}^{Imagen}	t_{rec}^{Imagen}	t_{pub}^{Imagen}
Promedio	Dentadura	208216	-	-	207985
Promedio	Coca	212648	-	-	212381
Promedio	Beso	209566	-	-	209392
Máximo	Dentadura	247087	-	-	246938
Máximo	Coca	233821	-	-	233378
Máximo	Beso	251937	-	-	251642
Mínimo	Dentadura	178599	-	-	178298
Mínimo	Coca	180606	-	-	180490
Mínimo	Beso	151246	-	-	151063

Tabla 6.32: Caso 1. Comparativa. (Valores en microsegundos)

La tabla 6.33 muestra los valores resumen del Caso 2. En este caso no se aportan los valores del reconocimiento de la imagen al no haber seleccionado la opción. Se puede observar el efecto del recorte en la imagen enviada en el tiempo de publicación de la imagen que al ser la recortado el coste es mucho menor que en los casos 1 y 3.

	Clase	t_{TOTAL}^{SAR}	t_{clp}^{Imagen}	t_{rec}^{Imagen}	t_{pub}^{Imagen}
Promedio	Dentadura	123879	123120	-	798
Promedio	Coca	191882	191176	-	682
Promedio	Beso	154056	153227	-	804
Máximo	Dentadura	146169	144911	-	1234
Máximo	Coca	236887	236401	-	1465
Máximo	Beso	171943	171274	-	1893
Mínimo	Dentadura	88316	87270	-	478
Mínimo	Coca	166207	164691	-	448
Mínimo	Beso	129465	127525	-	464

Tabla 6.33: Caso 2. Comparativa. (Valores en microsegundos)

La tabla 6.34 muestra los valores resumen del Caso 4. En este caso se aportan los valores del reconocimiento de la imagen y del recorte de la imagen enviándose la imagen reducida.

	Clase	t_{TOTAL}^{SAR}	t_{clp}^{Imagen}	t_{rec}^{Imagen}	t_{pub}^{Imagen}
Promedio	Dentadura	256617	125380	17315	640
Promedio	Coca	345111	214896	16406	961
Promedio	Beso	295884	160890	18042	857
Máximo	Dentadura	285343	146752	22044	1480
Máximo	Coca	374359	244314	20236	1822
Máximo	Beso	323616	182867	21995	1619
Mínimo	Dentadura	226018	98564	14412	455
Mínimo	Coca	302666	176243	14854	469
Mínimo	Beso	279224	145204	14870	446

Tabla 6.34: Caso 4. Comparativa. (Valores en microsegundos)

6.5.4. Tiempos de la toma de medida de tiempos

El tiempo total en el SAR t^{SAR}_{TOTAL} incluye la toma de medidas de cada una de las actividades que realiza. Se pretende determinar que tiempo se introduce en la medida total la instrumentación realizada. Para ello, se ha realizado un programa de evaluación del coste de las operaciones introducidas para la toma de medidas. El listado 6.5.4 muestra el esquema seguido para esta medida.

```
1   niter = 1000;
2   tini = chrono::steady_clock::now();
3
4   for (i=0; i < niter; i++) {
5     // codigo instrumentado para la toma de una muestra de tiempo
6       t1 = chrono::steady_clock::now();
7       t2 = chrono::steady_clock::now();
8       tdif = chrono::duration_cast<chrono::nanoseconds>(t2 - t1).count();
9     // fin del codigo intrumentado
10  }
11  tfin = chrono::steady_clock::now();
12  ttotal = chrono::duration_cast<chrono::nanoseconds>(tfin - tini).count();
13  tunit = ttotal / niter;
14  printf("T-Total: %d; T-unitario: %d ns; \n", ttotal,tunit);
```

El código comentado es el añadido en cada una de las operaciones de la toma de muestras. Concretamente para cada toma del tiempo total t^{SAR}_{TOTAL} en los casos mostrados de peor caso, se realizan 3 invocaciones de este código.

El programa se ha ejecutado 10 veces en la Raspberry y el resultado obtenido es el mostrado en la tabla 6.35.

No. Prueba	Tiempo de la medida
1	2553
2	2626
3	2634
4	2686
5	2550
6	2550
7	2624
8	2775
9	2550
10	2550
Promedio	**2609**
Máximo	**2775**
Mínimo	**2550**

Tabla 6.35: Medida del código de instrumentación. (Valores en nanosegundos)

Por lo que al Tiempo total del SAR t^{SAR}_{TOTAL} se le debería restar 3 veces el tiempo mínimo (7.65 μs) para eliminar la parte de la toma de medidas.

6.6 Conclusiones

En este capítulo se han realizado todas las pruebas para poder obtener los resultados finales con los que se desarrollarán las conclusiones finales del capítulo siguiente.

En este capítulo se han presentado el conjunto de experimentos realizados con el prototipo desarrollado y el conjunto de imágenes de golosinas que han sido utilizadas para las pruebas.

Las pruebas han estado encaminadas a analizar el proceso de generación de los modelos en ambos lenguajes utilizados con el fin de comparar sus prestaciones. Para ello se ha experimentado de forma incremental aumentando el número de clases a fin de estudiar las prestaciones con la cantidad de clases de objetos. La tabla resumen y las gráficas han mostrado que las diferencias entre ambas soluciones son similares.

Asimismo, se han presentado los resultados de los distintos tiempos totales y parciales de las distintas operaciones que se ejecutan en el SAR en función de las opciones elegidas en la interfaz del usuario en el SG.

Los resultados detallados de los tiempos en 3 de las clases de objetos ha permitido ofrecer un análisis de los tiempos peores, mejores y promedio obtenidos. Con estos datos extensibles con pequeñas variaciones al resto de objetos permite ofrecer una guía aproximada para la mejora de una versión más eficiente para su posible industrialización.

CAPÍTULO 7
Conclusiones

Este TFG se ha realizado con una acción de prácticas en empresa en el AI2 en el periodo de enero a septiembre de 2019 y se enmarca en las actividades del Instituto de Automática e Informática Industrial (AI2) y sus relaciones con las necesidades de las empresas que forman parte de sus contactos. Una de estas empresas, planteó una problemática que requería la identificación y etiquetado de objetos (golosinas) en bolsas con un número de unidades controladas.

De las problemáticas identificadas se escogió, por singularidad, una que requería identificar en una bolsa con productos comestibles (golosinas) el número de elementos de cada tipo que se habían envasado. La necesidad de que las bolsas contengan el número determinado de cada tipo de producto es fundamental para el cliente. Esto requiere la identificación de cada clase de objeto (golosina) y el contador del número de ellas para su empaquetado. El problema se aborda en su primera fase, un prototipo capaz de identificar y etiquetar las golosinas en una cinta transportadora. En otra fase posterior a este TFG ya se planteará el número de ellas por paquete.

Por lo tanto, en este trabajo de TFG se ha planteado la realización de un prototipo de sistema empotrado capaz de realizar la adquisición y el reconocimiento de objetos de una clase de objetos determinada. Los objetos, golosinas de las que se pueden obtener en muchos centros comerciales, representan un problema en su empaquetado en la empresa empaquetadora de estos productos. La solución abordada tiene como objeto la realización de un prototipo que, de tener los resultados que se han obtenido, se pueda industrializar siguiendo los estándares industriales apropiados.

El trabajo realizado en este TFG incorpora técnicas estudiadas y aprendidas durante los estudios de Ingeniería Informática cubriendo aspectos como desarrollo de sistemas empotrados, desarrollo de interfaces de usuario, redes de comunicaciones, desarrollo de software de sistema basado en técnicas de compartición de datos en sistemas distribuidos (middlewaare) y otros materias que han resultado básicas en el diseño y desarrollo del prototipo. Mención especial a estas técnicas son las propias de la ingeniería de desarrollo del software con sus fases de especificación, diseño, desarrollo y validación del producto.

En el capitulo 2 se ha planteado el estado de la técnica en aquellas tecnologías nucleares al TFG: sistemas empotrados, sistemas operativos, plataformas de sistemas empotrados comerciales y un análisis de las técnicas de inteligencia artificial y en especial de las redes neuronales.

En el capítulo 3 se ha diseñado el sistema prototipo a partir de los casos de uso planteados sobre el sistema global. El primer paso ha sido desarrollar los requisitos que han guiado el diseño arquitectónico y detallado del sistema.

En el capítulo 4, se han realizado las actividades propias del diseño del prototipo.La visión global y la descomposición del sistema en componentes interconectados con interfaces claramente identificadas. En este capítulo se han descrito las técnicas de aprendizaje profundo que han guiado el desarrollo.

En el capítulo 5 se ha planteado la implementación del sistema. Desafortunadamente, la cantidad de código resultante del prototipo excede cualquier tamaño de documento y, además, mucho de este código es convencional y típico de esta clase de aplicaciones. Se ha optado por detallar ciertas partes más significativas y obviar aquellas partes más convencionales.

Se han destacado dos soluciones basadas en redes neuronales que siendo básicamente muy iguales, representan visiones a evaluar. Por un lado el proceso de generación de un modelo de red neuronal de aprendizaje profundo desarrollado en un lenguaje como R que permite sobre herramientas como Keras elaborar un modelo.

Por otro lado, el uso del lenguaje Python para prácticamente lo mismo, generación del modelo. Python, a diferencia de R, es un lenguaje mucho mas extendido y su uso industrial mucho mas aceptado. La comparativa en términos de modelo para después calcular el grado de acierto y los tiempos de respuesta era uno d los objetivos del trabajo.

Finalmente, en el capítulo 6 se describe la experimentación realizada tanto en la fase de generación del modelo como de la fase de ejecución. Para la generación del modelo se ha optado por ha optado por paso a paso ir evaluando los modelos y, por lo tanto, la red neuronal subyacente. la experimentación ha partido de 2, 4,8 ,10, 12 y finalmente, todas clases de golosinas. En el capítulo se detallan los resultados obtenidos. En la fase de ejecución se ha propuesto el modelo temporal del tiempo de cómputo del proceso completo desde la petición de la imagen a la lectura de la imagen leída y la clase identificada. Se han han realizado un número significativo de pruebas para un subconjunto de las clases y se han extraído los tiempos te cómputo de cada una de las operaciones. Se han presentado las tablas con la comparativa de las distintas opciones que el usuario puede elegir en la interfaz del sistema de gestión (SG).

El capítulo 7, este, recoge las conclusiones del TFG. Dos conclusiones se pueden realizar, primero que se han alcanzado los objetivos técnicos planteados y se han detallado en este documento que considero que recoge todo el proceso de diseño y desarrollo. Una segunda conclusión va mas allá del trabajo realizado y es de carácter personal. este trabajo me ha permitido abordar un problema ingenieril, diseñar una solución y evaluarla aplicando las enseñanzas adquiridas en mis estudios. el resultado es altamente gratificante en lo personal y en lo técnico.

De forma global, se han cubierto los objetivos del trabajo:

- Se ha seleccionado y desarrollado un sistema empotrado que cubre las necesidades del desarrollo del sistema.
- Se ha seleccionado las técnicas de visión por computador para la identificación de los componentes.
- Se ha diseñado la arquitectura hardware y software que ha permitido la gestión del sistema de forma remota.
- Se ha utilizado un conjunto de piezas para entrenar y validar el modelo.
- Se ha diseñado y desarrollado las aplicaciones que han permitido la captura de imágenes, su uso para el entrenamiento de una red neuronal y la explotación del resultado de entrenamiento en el sistema empotrado.

- Se ha realizado la evaluación del sistema desarrollado

CAPÍTULO 8
Trabajos futuros

En este capítulo se exponen algunas de actividades a realizar en un futuro a medio y largo plazo. En primer lugar hay que tener en cuenta que este proyecto surge de las necesidades expresadas por una empresa en una petición de ayuda y soporte al Instituto de Automática Industrial AI2. A continuación se enumeran los posibles trabajos futuros.

El trabajo de futuro de referencia sería convertir el prototipo en una versión industrial robusta, eficiente y totalmente integrada en la cadena de producción de la empresa. Para conseguir este objetivo de futuro, se plantean una serie de acciones intermedias que se detallan a continuación.

- Mejorar la interfaz con el usuario para que se integre en un proceso automático. Identificar aquellas opciones que sean por defecto para un funcionamiento sin operador. Añadir un fichero de configuración de la interfaz que al iniciarse lea las opciones y adapte la interfaz en función de ellas.

- Integrar en la interfaz las herramientas que en el prototipo se ejecutan de forma separada como son la generación del modelo y la transformación para su integración en el sistema de adquisición y reconocimiento (SAR). Esto permitiría hacer transparente al usuario final estas actividades. Lo ideal sería conseguir hacer una transformación del modelo automática y evitar el proceso de *frugally-deep* que permite integrar como una librería en C++ que evita montar la aplicación C++ con el Tensor-Flow evitando código mucho mas grande que requiere mayor tiempo de cómputo.

- Adecuar el sistema para su funcionamiento integral continuo. En este punto, sería importante que el sistema pudiese identificar nuevas clases de objetos y realizar el aprendizaje de forma automática.

- Este prototipo ha sido orientado a la adquisición y reconocimiento de piezas aisladas. Sería fundamental que fuese capaz de reconocer escenarios con varias piezas y capacidad para contarlas en bloques diferenciados con el fin de que su empaquetado incluyese un número de golosinas determinado de cada clase de ellas. Con la base establecida de recorte de la pieza y su aislamiento, no sería complicado abordar esta problemática.

- Aunque las redes neuronales utilizadas en el prototipo han demostrado un nivel alto de eficiencia y exactitud, una acción de futuro es evaluar otras técnicas para analizar su comportamiento, su precisión y tiempo de respuesta. En este sentido, otro tipo de redes, por ejemplo las Redes Generativas Antagónicas GAN podrían ser una alternativa a considerar.

- Respecto al hardware, el sistema empotrado utilizado es altamente fiable, robusto y de bajo coste. Pero existen otras alternativas que podrían ser utilizadas. La disponibilidad de un sistema operativo como Linux, permite sin mucho trabajo, portar el trabajo de un sistema empotrado a otro.
- Los tiempos de reconocimiento no son un factor limitante. Resulta extremadamente rápido el sistema como para que esto sea un obstáculo. No obstante habría que considerar soluciones tipo hardware. En ese sentido, existen placas que incluyen FPGA en la propio hardware de la placa. Un trabajo de futuro podría centrarse en pasar el modelo de ejecución para el reconocimiento, o sea la red neuronal, a un modelo basado en VHDL para la programación en la FPGA. Esto requeriría usar la interfaz del software programado en la FPGA para el reconocimiento y clasificación del objeto.

Algunos de estos trabajos se podrán abordar a corto plazo. Sobre todo aquellos de evaluación de alternativas, introducción de nuevas técnicas, programación en la FPGA, etc. Otros, la solución es a largo plazo (1 año mínimo) como el proceso ingenieril asociado al desarrollo del producto industrial. Esto supone, revisar los casos de uso para adecuarlos a las necesidades del cliente, redefinir los requisitos de base, desarrollar los requisitos técnicos, realizar un diseño completo (arquitectónico y detallado) e implementar la solución reescribiendo el código para cumplir los estándares industriales. Junto a ello, el proceso de verificación y validación sería obligatorio para completar el todo el desarrollo del software.

Glosario

8.1 Acrónimos

Los acrónimos utilizados en este TFG se listan a continuación con su significado.

Acrónimo	Significado
AI2	Instituto de Automática e Informática Industrial
API	Interfaz de Programación de Aplicaciones
BN	Blanco y negro
CNN	Convolutional neural networks.
CPU	Unidad Central de Proceso
CUDA	Compute Unified Device Architecture
DBN	Deep Belief Networks
DCPS	Data-Centric Publish and Subscribe
DDS	Data Distribution Service
FPGA	field-programmable gate array
FPU	Floating Point Unit
GAN	Redes Generativas Antagónicas
GPIO	General Purpose Input Output
GPU	Graphic Processing Unit
GUID	Globally Unique Identifier
HDF5	Hierarchical Data Format version 5
IoT	Internet of Things
JSON	JavaScript Object Notation
LCD	Pantalla de cristal líquido
MOM	Middleware orientado a mensajes (MOM)
MQTT	Mensaje Queue Server telemetry transport
OMG	Object Management Group
OPC	OLE for Process Control
PMOD	Peripheral Module interface
RAM	Random Access Memory
RB	Requisitos de base
ReLu	Rectifier Linear Unit
RGB	Red, green and blue
RN	Redes neuronales
RNN	Recurrent neural networks

Acrónimo	Significado
RPC	Remote Procedure Call
SAR	Sistema de adquisición y reconocimiento
SG	Sistema de gestión
SGM	Sistema de generación del modelo
SSD	Unidad de estado sólido
TFG	Trabajo Fin de Grado
TPU	Tensor Processing Unit
VHDL	Very high level Hardware Description Language

Tabla 8.1: Acronimos

Bibliografía

[Ber18] Fernando Berzal. *Redes Neuronales & Deep Learning*. Edición independiente, 2018.

[BG14] Andrew Banks and Rahul Gupta. Mqtt version 3.1.1., 29 October 2014.

[CA06] Alfons Crespo and Alejandro Alonso. Una panorámica de los sistemas de tiempo real. *Revista iberoamericana de automatica e informatica industrial (RIAI), ISSN 1697-7912, Vol. 3, Nº. 2, 2006, pags. 7-18*, 4, 01 2006.

[Fou15] OPC Foundation. Unified architecture specification, 2015.

[Gro15] OMG Group. Data distribution service (dds), April 2015.

[Her18] Tobias Hermann. Frugally-deep tool. a keras model transformer to c++. https://github.com/Dobiasd/frugally-deep, 2018.

[Loz18] Andrés Murgui Lozano. Clasificación y reconocimiento de imágenes con redes neuronales para entornos industriales., 2018.

[OPT09] Fundación OPTI. Tendencias y aplicaciones de los sistemas embebidos en españa., 2009.

[Pi17] Raspberry Pi. Linux distribution for raspberry pi, 2017.

[Sim18] José Simó. Modelos para el diseño e implementación del software intermediario en el desarrollo de sistemas de control., Noviembre, 2018.