Universidad de La Laguna

Escuela Superior de Ingeniería y Tecnología
Sección de Ingeniería Informática

"Deep Learning" para reconocimiento de imágenes en Raspberry Pi 2

Deep Learning for image recognition in Raspberry Pi 2

Rodrigo Colombo Vlaeminch

La Laguna, 4 de marzo de 2016

D. Jose Demetrio Piñeiro Vera, con N.I.F. 43774048-B profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Deep Learning para reconocimiento de imágenes en Raspberry Pi 2”

ha sido realizada bajo su dirección por D. **Rodrigo Colombo Vlaeminch**,

con N.I.F. X5997690-A.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de marzo de 2016

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor Jose Demetrio Piñeiro Vera por su gran ayuda y apoyo durante la elaboración del proyecto.

A mis padres y mi hermano, que me dieron fuerza y ejemplo para realizar mis estudios.

A mis amigos, tanto de la carrera como de fuera, que proporcionan esas horas de calma y relax que siempre hacen falta.

Por último, y no por ello menos importante, a mi pareja, por aguantar mi estrés y mal humor animándome en todo momento.

A todos, muchas gracias.

Licencia

Resumen

El objetivo de este trabajo ha sido explorar una aplicación concreta de las técnicas de Deep Learning a un ordenador de pequeño formato y capacidad computacional limitada como la Raspberry Pi. Esta es un pequeño ordenador con buenas prestaciones y muy bajo coste. Las técnicas de Deep Learning, o de redes neuronales profundas, se encuentran en auge y aparecen en numerosas aplicaciones pero requieren normalmente una elevada capacidad de cómputo. Así, finalmente, se desarrolló a modo de prueba de concepto una aplicación simple de reconocimiento de imágenes basada en Deep Learning con las capacidades que dicha placa nos presta.

Palabras clave: Deep Learning, Raspberry Pi, redes neuronales, reconocimiento de imágenes.

Abstract

The main goal of this project was to explore a particular application of Deep Learning techniques in a small format, limited computational power computer such as the Raspberry Pi. This is a credit-card sized computer with good features and low cost. Deep Learning techniques, or deep neural networks, are very popular and used in many applications, but usually require considerable computational power. As a proof of concept, a simple application for image recognition based on Deep Learning was developed, exploiting the computational resources of the Raspberry Pi.

Keywords: Deep Learning, Raspberry Pi, neural networks, image recognition.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

En este capítulo se hace una introducción sobre el planteamiento del proyecto y las herramientas utilizadas.

1.1 Planteamiento del Proyecto

El objetivo de este proyecto trata de explorar si en una plataforma computacionalmente limitada como la Raspberry Pi era posible ejecutar alguna aplicación basada en las técnicas de Deep Learning.

Las ventajas de un sistema así serían muchas, dado su pequeño tamaño y consumo. Se podría diseñar un sensor inteligente que captara una situación compleja, como por ejemplo, reconocer un objeto o persona por su imagen, o comprender una orden de voz, apoyándose en las capacidades de reconocimiento de patrones de esas redes.

Como desventaja, se sabe que los algoritmos de Deep Learning requieren grandes capacidades computacionales, al menos en su fase de aprendizaje o entrenamiento, y posiblemente también, una vez entrenados, en su uso convencional como detectores de patrones.

Por otro lado, la Raspberry Pi dispone de ciertos recursos como su procesador gráfico VideoCore o en el caso de la Raspberry Pi 2, de su conjunto de instrucciones Neon de tipo vectorial, que podrían adaptarse a los requisitos de implementar las funciones de una red o integrarse en una librería de desarrollo de aplicaciones de Deep Learning.

Así se pretende demostrar que no es necesario disponer de un gran ordenador para hacer los cálculos que realiza el Deep Learning.

Con esto se intenta enviar el mensaje de que no hace falta tener grandes recursos para llevar a cabo proyectos de gran interés y que puedan proporcionar soluciones a muchos problemas.

1.2 Raspberry Pi 2 B

Se trata de una pequeña placa base u ordenador de placa única (SBC), desarrollado en el Reino Unido por la Fundación Raspberry Pi. Este pequeño aspirante a PC se desarrolló con fines académicos, para estimular la enseñanza de ciencias de la computación, es por ello que tiene un precio muy económico (35USD) y es de uso libre tanto a nivel educativo como particular, aunque se trata de un producto de propiedad registrada.

1.2.1 Hardware y componentes

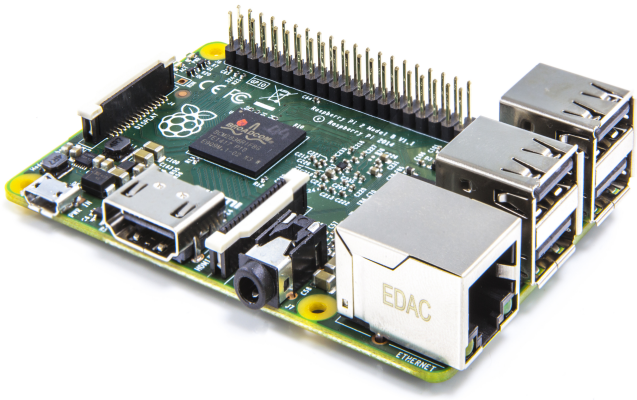

Figura 1.1: Raspberry Pi 2

La Raspberry Pi 2 B consta de:

- Un SoC Broadcom BCM2836 con una CPU de 4 núcleos ARM Cortex-A7 de 900MHz.
- 1GB de RAM LPDDR2 SDRAM de 450MHz.
- Procesador gráfico Broadcom VideoCore IV.
- Una ranura microSD, donde se almacena su sistema operativo.
- 4 USBs 2.0.
- Un puerto Ethernet de 10/100Mbps.
- Un puerto HDMI Full HD.
- Una interfaz DSI para pantalla LCD.
- 40 pines GPIO.

1.2.2 Sistemas operativos

Además del hardware enumerado en el punto anterior, podemos instalar distintos sistemas operativos y distribuciones en la placa, de este modo podremos crear un buen entorno de desarrollo, o darle un uso adecuado a nuestras necesidades. Entre los numerosos SO que podemos encontrar, cabe destacar:

- Raspbian: basado en Debian, está hecho específicamente para la CPU ARMv6 de la Raspberry Pi, con soporte por hardware para cálculos en punto flotante. Esta distribución está orientada especialmente para la enseñanza de informática, por lo que cuenta con herramientas de desarrollo como Scratch o IDLE. A su vez, al tratarse de un GNU/Linux se pueden descargar multitud de programas y aplicaciones como si se tratase de un sistema de escritorio convencional.

- Ubuntu MATE: es una distribución basada en Ubuntu con el entorno de escritorio MATE, que a su vez es un derivado del entorno Gnome 2. Al igual que el anterior, es un Linux, por lo que tenemos infinidad de posibilidades de uso.

- Windows CE: sistema operativo desarrollado por Microsoft para sistemas embebidos, al ser compatible con los procesadores ARM, se puede instalar en la Raspberry Pi.

- OpenELEC: es una distribución de propósito único, desarrollado como Centro Multimedia, ya que proporciona numerosas aplicaciones para esta finalidad. Es una sistema GNU/Linux de arranque muy rápido y extremadamente ligero, pensado especialmente para arrancar desde tarjetas Flash o SSD.

Estos son algunos de los sistemas que se pueden instalar en la Raspberry Pi 2, para este trabajo se utilizó el Raspbian, ya que cumple con los requerimientos necesarios para la implementación de nuestro proyecto y es el recomendado por la fundación Raspberry Pi.

1.3 Deep Learning

Figura 1.2: Deep learning

1.3.1 Definición

El Deep Learning, o aprendizaje profundo, es una familia de algoritmos que simulan el proceso realizado por las neuronas cerebrales para llevar a cabo el reconocimiento de voz, imágenes o palabras. Dichos algoritmos funcionan en un proceso por capas, y en cada capa encontramos neuronas que realizan un procesamiento simple sobre las salidas de la capa de neuronas anterior. Sin embargo, no conviene llevar demasiado lejos la analogía con las estructuras que podemos encontrar en un cerebro real, ya que las neuronas artificiales son mucho más simples que las naturales. Podemos asumir que las redes neuronales artificiales constituyen modelos matemáticos, con parecido superficial a las redes naturales, que realizan funciones útiles porque pueden adaptarse o aprender en cierto sentido.

Otra definición de Deep Learning podría ser que se trata de un procesador de información que recibe entradas, codificadas como números, y después de pasar por una serie de operaciones matemáticas (capas de neuronas) produce información de salida codificada también y que podemos adaptar para que realice una función deseada concreta.

1.3.2 Funcionamiento

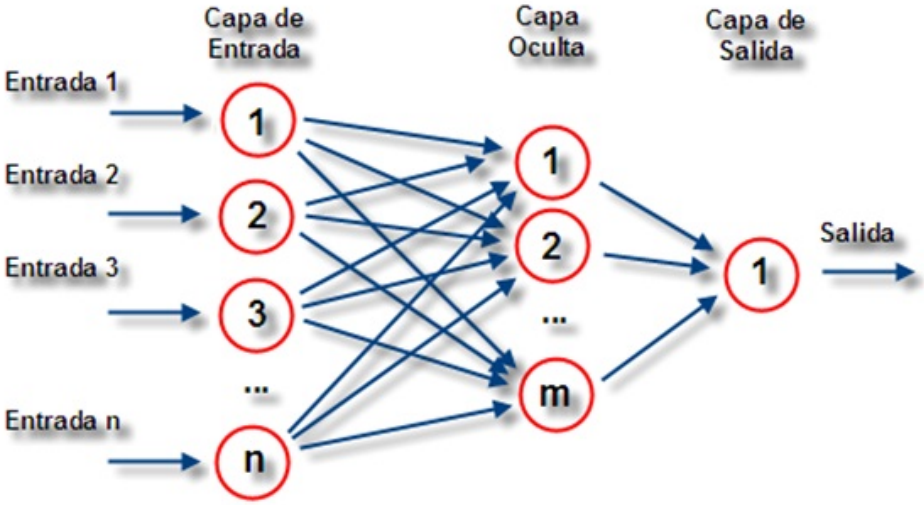

Figura 1.3: Red neuronal

Para explicar el funcionamiento de una red neuronal clásica, nos basaremos en la Figura 1.3. Las neuronas se organizan en capas, donde cada una se conecta con todas las de la capa siguiente. Cada conexión tiene asociado un peso, por lo que, la principal operación que se realiza es una multiplicación entre el valor de la neurona y su conexión saliente. Las neuronas de las capas siguientes reciben los resultados anteriores sumados en uno y aplican una función no lineal para producir un nuevo resultado. Existen numerosas funciones, aunque la más común es la función sigmoide [Figura 1.4].Una de sus características es que su salida se puede interpretar como una probabilidad, ya que da un valor en el rango [0, 1].

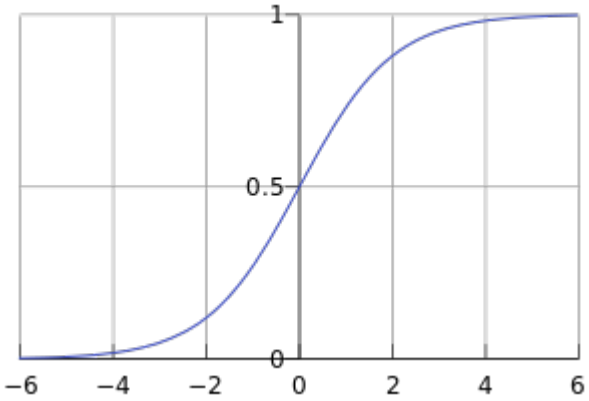

Figura 1.4: Función Sigmoide

El proceso de “aprendizaje” de una red neuronal de este tipo consiste en hacer que la red “aprenda” la relación entre una serie

de entradas y las salidas deseadas para cada una de esas entradas. Es decir, se supone que a partir de una serie de ejemplos de lo que queremos que la red realice, ésta pueda generalizar o deducir la relación que existe entre entradas y salidas. Este proceso de aprendizaje se llama entrenamiento de la red y consiste en modificar la parte adaptable de la misma, que son los valores de los pesos, hasta conseguir el comportamiento deseado.

1.3.3 Tipos de Redes Neuronales

Existen diferentes arquitecturas de redes, entre ellas podemos destacar:

- Redes de neuronas en capas completamente conectadas (perceptrón multicapa): es la organización de la red neuronal clásica. Se organizan en capas, y sólo se permiten conexiones de las salidas de una capa a las entradas de la capa siguiente.
- Redes de neuronas recurrentes: no se organizan en capas, sino que tienen conexiones arbitrarias entre todas, incluso pueden crear ciclos. Con los ciclos permitimos que la red tenga memoria, por lo que las entradas introducidas en un momento dado pueden seguir circulando por la red aunque hayamos añadido nuevas entradas.
- Redes convolucionales: esta red sí se organiza en capas, aunque ahora no todas las neuronas están conectadas con las de la capa siguiente, en este caso solo se conectan con algunas, de este modo las neuronas se especializan en una región de la capa anterior, así se reduce el número de pesos y multiplicaciones que se deben realizar.

  Para este proyecto se utilizó la red descrita por Alex Krizhevsky, Ilya Sutskever y Geoffrey Hinton [1], la cual consta de ocho capas entrenables: tres totalmente conectadas (no recurrentes) y cinco convolucionales.

Capítulo 2
Estudio previo

En este capítulo se describe el estudio realizado con el objetivo de ver la situación del tema a tratar para realizar el proyecto.

2.1 Estado del arte

A pesar de que el Deep Learning parece una técnica nueva e innovadora, realmente es un tema en el que se lleva investigando más de 30 años. En el año 2012 un grupo de investigación de la Universidad de Toronto obtuvo unos resultados sorprendentes en la ILSVRC (ImageNet Large Scale Visual Recognition Challenge), la principal competición de reconocimiento de imágenes.

Desde entonces este tipo de técnicas a tomado un importante papel en muchas áreas de la inteligencia artificial. Por ello, empresas como Google, IBM, Facebook están invirtiendo en estas técnicas y contratando a los mejores investigadores. En muchas otras competiciones de aprendizaje automático, las técnicas de Deep Learning superan a técnicas más tradicionales en todo aquello que tiene que ver con el reconocimiento de patrones, como procesamiento de escritura, diversos tipos de reconocimiento de imágenes, voz, etc.

Existen numerosas aplicaciones que utilizan este tipo de técnicas, Siri de Apple usa Deep Learning para el procesamiento de lenguaje natural, al igual lo hace Google para el reconocimiento de voz en los dispositivos Android y para buscar imágenes en Google+ por el contenido, sin estar éstas etiquetadas.

En cuanto a la Raspberry Pi 2, existe una gran variedad de aplicaciones y usos, las más usuales son:

- Centro multimedia, con el cual podemos ver películas, series, televisión, escuchar música, etc.

Figura 2.1: Centro multimedia

– Consola de juegos, con la Raspberry Pi y algún elemento externo, como puede ser un Joystick, pulsadores, una pantalla, etc.

Figura 2.2: Consola Pi

– Uso como ordenador convencional, simplemente conectando un teclado, un ratón y un monitor tendremos un ordenador.

– Aplicaciones de domótica, gracias a sus pines GPIO podemos conectar sensores, relés, leds, etc. por lo que se puede utilizar con este fin.

Como vemos, el Deep Learning no está presente en estas aplicaciones y debido a la gran proyección de futuro que presenta este ámbito y a lo poco que se ha aplicado en dispositivos de placa única (SoC), este proyecto se centró en sacar partido de la Raspberry Pi 2 con un desarrollo de este tipo.

2.2 VideoCore IV

En febrero de 2014 Broadcom liberó el código fuente del controlador encargado de los gráficos del VideoCore IV. Como mencionamos anteriormente, la Raspberry Pi lo tiene integrado en su circuito SoC Broadcom BCM2836.

Dada la noticia, nos planteamos utilizar el Videocore como

herramienta para realizar los cálculos que lleva el Deep Learning, ya que con esta liberación se puede utilizar la GPU para más tareas que procesar gráficos.

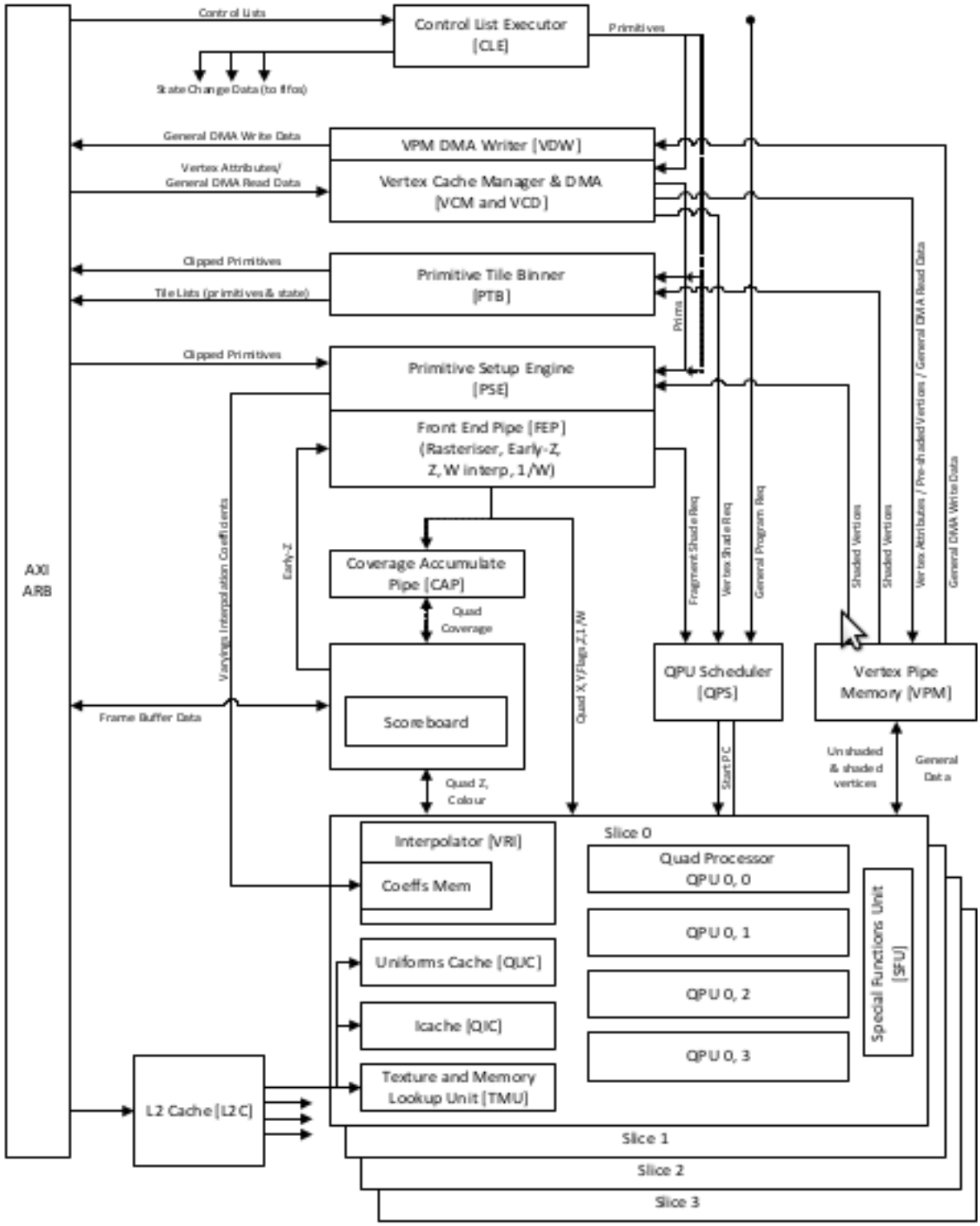

Figura 2.3: Estructura VideoCore IV

Como podemos ver en la imagen contamos con 12 núcleos

(QPUs), por lo que podemos pensar que tenemos una gran potencia a nuestra disposición. Sin embargo no hay demasiada memoria RAM, solo disponemos de 4096 Bytes que corresponden a la Vertex Program Memory.

Broadcom también publicó un manual [8] con una buena descripción de la interfaz hardware, pero solo con esto no es suficiente para ejecutar código en la GPU, haría falta implementar herramientas programando directamente a bajo nivel, ya que no existe ninguna API (interfaz de programación de aplicaciones) como pueden ser CUDA o RenderScript, tarea que dificulta mucho el avance del proyecto.

Puesta en marcha la búsqueda de información para realizar este proyecto, la fundación Raspberry Pi lanzó al mercado el modelo Pi 2 B, con la cual el uso de la CPU para los cálculos no suponen un problema. De modo que, evaluadas las ventajas e inconvenientes, quedó descartada la opción de programar a bajo nivel para optimizar el Deep Learning en el VideoCore.

2.3 Proyecto DeepBeliefSDK

Este proyecto fue desarrollado por Pete Warden con el objetivo de darle a los dispositivos móviles la capacidad de ver y ser base para el desarrollo de aplicaciones de reconocimiento de imágenes. Es una implementación en una librería de la arquitectura de red neuronal descrita por Alex Krizhevsky, Ilya Sutskever y Geoffrey Hinton [1].

El código está altamente optimizado para funcionar en aparatos con limitaciones de memoria y procesamiento, tales como Smartphones, Raspberry Pi, Beaglebone Black; aunque también se puede utilizar en ordenadores de sobremesa, con OpenCV, en Javascript.

El proyecto ha sido probado en un iPhone 4, un Samsung Galaxy S5, donde se obtuvieron unos resultados de alrededor de 650ms para la clasificación; en PCs de sobremesa con Ubuntu 12.04 y 14.04 en plataformas x86-64, en PCs con OS X, en la Raspberry Pi 1, con la cuál se obtuvieron unos resultados de 5 seg para el reconocimiento y 3 seg con overclock; en la Raspberry Pi 2, con un rendimiento de 3,8 seg y con overclock 3,2 seg.

Además de los dispositivos anteriores, el proyecto se puede utilizar en cualquier otra plataforma, siempre y cuando tenga disponible un compilador y las bibliotecas estándar de C, ya que está diseñado para no tener ninguna dependencia por defecto, exceptuando las del lenguaje C, código en el que está programado.

DeepBeliefSDK [2] también consta con códigos de ejemplo para la ejecución de reconocimiento de imágenes en las distintas plataformas anteriormente nombradas, funciones para poder llamar a las redes ya entrenadas o entrenar la última capa con nuestras propias imágenes.

A su vez tenemos tres redes que podemos utilizar:

- jetpac.ntwk: red entrenada por el propio Pete Warden con objetos de 999 tipos diferentes.
- ccv2010.ntwk y ccv2012.ntwk: redes que se construyeron en el proyecto libccv [9] entrenadas para la tarea de clasificación de imágenes, es decir, predecir que objeto u objetos se encuentran en una imagen determinada de entre una serie de categorías predeterminadas. En este caso, entrenadas partiendo de imágenes con 1000 clases de objetos diferentes etiquetados.

Capítulo 3
Desarrollo

En este capítulo se describe las herramientas necesarias para el desarrollo llevado a cabo para completar el proyecto.

3.1 Herramientas necesarias

Para la realización de este proyecto se necesitaron algunos componentes y recursos básicos:

- Raspberry Pi camera, la cuál nos sirve para capturar imágenes en tiempo real para posteriormente aplicar el algoritmo de reconocimiento.

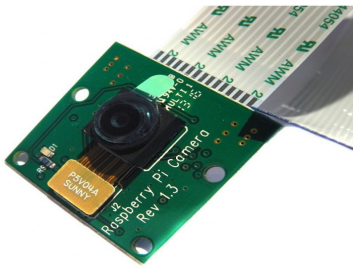

Figura 3.1: Pi Camera

- Pantalla TFT 2.8” con la que visualizamos las capturas tomadas y vemos el menú de trabajo o interfaz de usuario, a su vez consta de dos pulsadores que usamos para desplazarnos en el menú.

Figura 3.2: TFT 2.8"

- Conexión por SSH desde un PC hasta la Raspberry Pi para poder programar la aplicación directamente en la placa.
- Como resulta obvio, la Raspberry Pi, donde se llevó a cabo todo el proyecto.
- Librerías de Python “picamera” y “RPi.GPIO”, con las que controlamos la cámara y los pulsadores.
- Programa fbi (linux **f**rame**b**uffer **i**mageviewer) para realizar los envíos de las imágenes a la pantalla TFT.

3.2 Reconocimiento de imágenes

Para llevar a cabo el proceso de reconocimiento de imágenes utilizamos DeepBeliefSDK [2], la herramienta desarrollada por Pete Warden aunque solo la parte relacionada con la Raspberry Pi 2.

A diferencia del uso de dicha librería en la Raspberry Pi 1, donde la mayoría de cálculos matriciales que necesita la red neuronal se realizan en la GPU (VideoCore), en el modelo 2 se pasa a utilizar la CPU. Esto se debe a que, mientras el VideoCore sigue siendo el mismo, el nuevo procesador ARMv7 ha mejorado bastante e incluye nuevas instrucciones.

Sin embargo, el uso de la CPU no es impedimento para realizar el proyecto, puesto que no disminuye el rendimiento y además nos permite utilizar el VideoCore para la visualización de la interfaz de usuario en la pantalla TFT.

El motivo por el que el rendimiento entre las dos placas sea muy similar, aunque utilicemos componentes muy distintos, es que el nuevo procesador ARMv7 cuenta con 4 núcleos que son optimizados además con las instrucciones NEON de tipo vectorial, frente al ARMv6 de un sólo núcleo de la Raspberry Pi original. El reloj del procesador también ha pasado de 700 MHz a 900 Mhz.

Para sacar provecho de estas instrucciones se utiliza a su vez la librería de C++ EIGEN [3], con la que se optimizan todos los cálculos de álgebra lineal, utilizando la vectorización explícita para las instrucciones NEON. Esta librería no tiene ninguna dependencia, exceptuando las bibliotecas estándar de C++ y las instrucciones NEON en la compilación para ARM.

3.3 Rutinas GEMM

La optimización de los cálculos de Deep Learning en la librería DeepBeliefSDK se basa en usar una implementación muy eficiente de las rutinas GEMM (General Matrix to Matrix Multiplication) en

cada plataforma. Estas rutinas tratan esencialmente de multiplicar matrices de una forma eficiente sin aprovechar ninguna estructura especial en las matrices.

Para esto, en la memoria RAM las matrices deben almacenarse de forma consecutiva, ya sea fila a fila, o columna a columna; es decir, convirtiendo la matriz en un vector.

Estas rutinas nacen como parte de la biblioteca BLAS (Basic Linear Algebra Subrutines), creadas en 1979 para el lenguaje FORTRAN.

La importancia de las rutinas GEMM en el Deep Learning se debe a que las multiplicaciones de matrices ocupan entre el 89 y el 95% del tiempo que necesitan las capas de la red para el reconocimiento.

Esto se debe a que una sola capa de una red típica puede requerir la multiplicación de una matriz de 256 X 1152 y otra de 1152 X 192, lo que lleva un total de 57 millones de operaciones en punto flotante.

Para este proyecto, como mencionamos antes, utilizamos la biblioteca EIGEN [3] que implementa sus propias rutinas GEMM.

3.4 Instrucciones NEON

Son instrucciones de propósito general SIMD (Single Instruction, Multiple Data) de ARM, pensadas para procesar eficientemente formatos multimedia, tales como decodificación de vídeo, procesado de gráficos 2D y 3D, procesamiento del habla, síntesis de sonido, etc.

Estas instrucciones son una extensión de los procesadores ARM Cortex, como mencionamos anteriormente la Raspberry Pi 2 lleva el procesador ARM Cortex A7, por lo que se aprovechan dichas instrucciones.

Algunas características y formas de funcionamiento:

- Los registros se consideran vectores de elementos del mismo tipo de datos.
- Los tipos de datos pueden ser signed/unsigned de 8, 16, 32 o 64 bits y float con precisión simple.
- Como son del tipo SIMD, se realiza la misma operación a todos los carriles.

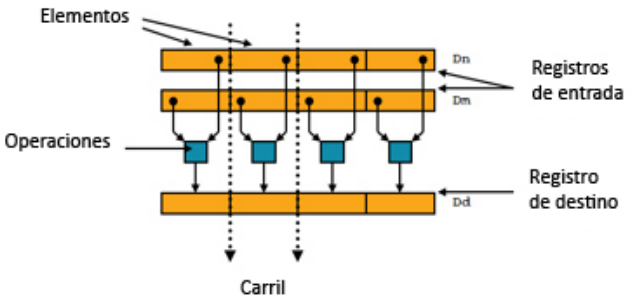

Figura 3.3: Funcionamiento NEON

– Consta de un archivo de registro con doble vista de 128bits/64bits que permite un manejo eficiente de los datos y minimiza los accesos a memoria.

Con estas instrucciones, el reconocimiento de imágenes se puede realizar relativamente rápido, ya que todas, o casi todas, las operaciones matriciales son atacadas desde aquí con la librería EIGEN [3].

3.5 Entrenamiento

La capacidad de procesamiento para entrenar una red con las dimensiones adecuadas para la base de datos ImageNet está hoy por hoy fuera del alcance de un dispositivo como la Raspberry Pi.

Como dato, simplemente el tamaño del conjunto de imágenes de esta base de datos es de unos 160 GB, existiendo unas 50 imágenes RGB de un tamaño variable por cada una de las 1000 categorías de objeto. La necesidad de cómputo para el entrenamiento es muy elevada.

Sin embargo, apoyándose en las ideas descritas en DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition [6], es posible considerar a una red ya entrenada (por ejemplo, resolviendo Imagenet con prestaciones adecuadas) como un preprocesador que analiza imágenes extrayendo de ellas características abstractas de alto nivel.

Como se considera en el artículo anterior, se pueden obtener las salidas intermedias de dicha red y conectarlas con una última etapa clasificadora simple. Esta última etapa sí podría ser entrenada de forma aislada para reconocer una nueva categoría de objeto con un coste computacional mucho más reducido. En DeepBeliefSDK [2] se contempla esta posibilidad, permitiendo entrenar un clasificador de tipo SVM (Support Vector Machine, usando la librería libSVM [7]) como última etapa, conectado con la salida de una de las últimas capas de la red.

Este tipo de clasificador necesita para su funcionamiento disponer de ejemplos positivos y negativos de la categoría del objeto a entrenar. Como ejemplo, si se utiliza la antepenúltima capa de la red antes citada, se obtiene un vector de 4096 salidas cuando ponemos una imagen a su entrada. El clasificador SVM toma esos 4096 números y la información de que la imagen de entrada es un ejemplar positivo o negativo para adaptarse. Cuando se presenta un número suficiente de ejemplares positivos y negativos, se obtiene una nueva red global capaz de detectar en una imagen la presencia o no de un objeto de la categoría entrenada.

En este trabajo, se probó a entrenar un objeto no presente en el entrenamiento de las redes mencionadas antes (no presente en Imagenet). Se trata de un rotulador de pizarra, de diversas formas y colores. Los entrenamientos se realizaron en un ordenador de sobremesa para acelerar el proceso.

Para entrenar la red se utilizó un total de 3788 imágenes, es decir 1894 ejemplos positivos y 1894 ejemplos negativos. De este total se utilizaro ¾ de las imágenes para el entrenamiento y ¼ para el test.

Figura 3.4: Ejemplo positivo

Figura 3.5: Ejemplo negativo

El tamaño de las imágenes no debería tener mayor influencia, ya que en primer lugar la red recorta las imágenes de entrada al cuadrado mayor posible para seguidamente reescalarlas a 256 x 256.

La red se centra en un subcuadrado de 224 x 224, que se posiciona según el parámetro “flags” que puede tomar los siguientes valores: si el parámetro es 0 se escoge centrado, dejando 16 px por cada lado; si es JPCNN_RANDOM_SAMPLE se escoge aleatoriamente; y si es JPCNN_MULTISAMPLE se eligen 10 posiciones diferentes y se lanzan las 10 a clasificar.

3.6 Interfaz

Para poder utilizar la aplicación y que al usuario le fuera sencillo, se desarrolló una interfaz de usuario en lenguaje Python.

La interfaz consiste en un menú en el que nos vamos desplazando con uno de los pulsadores, y unas acciones a realizar.

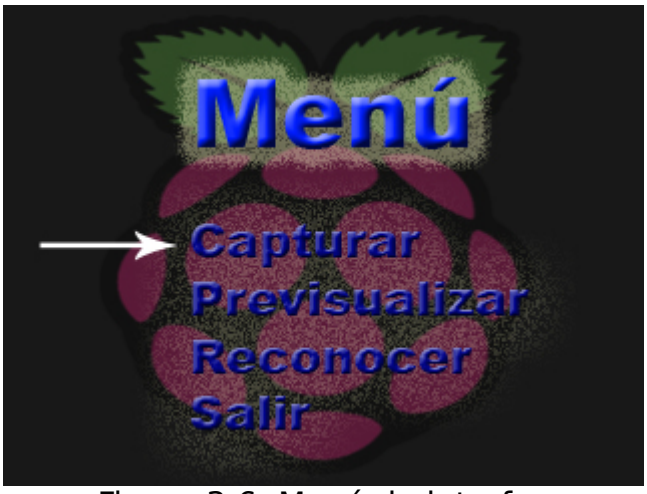

Figura 3.6: Menú de interfaz

Como podemos apreciar tenemos cuatro acciones a realizar.

3.6.1 Acción capturar

Con la acción capturar sacamos la foto, la almacenamos y la mostramos durante 3 segundos en la pantalla.

Para esto utilizamos la librería “picamera” implementada por Dave Jones con la que, cambiamos la resolución por defecto para poder adaptar la imagen a la pantalla TFT (320, 240) y realizamos la captura. Para el algoritmo de reconocimiento no es necesario escalar la imagen, ya que no importa el tamaño, todas se reescalan en el código a 256 X 256px.

Para enviar la captura a la pantalla hacemos uso del comando mencionado anteriormente “fbi” (Linux Frame Buffer Image).

Uno de los problemas encontrados a la hora de visualizar la imagen fue añadir el módulo en el kernel para controlar nuestra pantalla TFT. Por defecto, la Raspberry Pi 2 no lo trae, pero gracias a Notro [4], tenemos un desarrollo ya hecho de un módulo para controlarla.

Una vez descargado el módulo frame buffer tft “fbtft”, lo añadimos al kernel y recompilamos.

Recompilado el kernel, encontramos un nuevo dispositivo en /dev/fb1 al cual le podemos enviar las imágenes.

El funcionamiento de la acción capturar lo vemos reflejado en los métodos “capturar ()” y “envia foto (foto)” en el código que encontramos en el apéndice 1 – 7.1

3.6.2 Acción previsualizar

Esta opción del menú simplemente nos enseña durante 4 segundos la foto que tenemos almacenada, en caso de no encontrar imagen, se imprime un error en la pantalla.

Para esta tarea utilizamos la función “envia_foto (foto)”.

3.6.3 Acción reconocer

Se trata de la acción principal del programa, en este momento es donde se realiza el reconocimiento de la imagen capturada.

En primer lugar realizamos una llamada al programa jpcnn pasándole como argumento la foto capturada y la red neuronal a utilizar. Esta llamada escribe en un fichero los resultados obtenidos. Todo esto lo hacemos invocando a la función “reconoce (foto)”.

Una vez tenemos el fichero resultado, con la llamada a la función “resultado (fichero)” ordenamos el fichero con los datos, dejando en la primera línea el resultado con la probabilidad más alta.

Al leer la línea aplicamos el filtro de que la probabilidad sea mayor que 0.05, si esto se cumple imprimimos en la pantalla el resultado obtenido. En caso de no cumplirse, se muestra un error al reconocer la imagen.

3.6.4 Acción salir

Con esta acción limpiamos el programa, es decir, los ficheros creados y la foto tomada se borran de la memoria, y se limpian los pines utilizados con la función “GPIO.clenup ()”. Además, al ser este el objetivo de uso de la Raspberry Pi, una vez finalizada la ejecución apagamos el sistema.

3.7 Demonio

Para no tener que conectarnos vía SSH para ejecutar el programa, ni conectar un monitor para ver la consola, puesto que si visualizamos el entorno gráfico, cuando enviamos una foto a la pantalla TFT el proceso que ejecuta las X muere, se implementó un servicio que ubicamos en “/etc/init.d/” para que se ejecute al encender la placa.

La implementación del servicio podemos verla en el apéndice 1 – 7.2, con el cual realizamos una llamada al programa implementado anteriormente “interfaz.py”.

Al demonizar la interfaz surgió el problema de que desaparecía el proceso cuando seleccionábamos la acción capturar, esto es debido a que no encontraba las rutas de los ficheros ni las imágenes, puesto que al ejecutar el demonio nos situamos en el path del

usuario root.

Para solucionar este error, en la implementación de la interfaz se le pasan las rutas completas de los ficheros. Esto presenta un problema en la portabilidad del programa, ya que para poder ejecutarlo en otra placa debemos cambiar las rutas de los ficheros.

3.8 Funcionamiento e instalación

El proyecto lo podemos encontrar con todas sus dependencias en Github [5], para ejecutarlo en la Raspberry Pi 2 y, si lo deseamos, realizando algunas modificaciones podremos ejecutarlo en la Raspberry Pi 1 o en cualquier sistema Linux, siempre y cuando se cumplan las dependencias.

Las dependencias para nuestro caso son: la biblioteca EIGEN, C++ y Python 2.7.

En primer lugar debemos instalar las librerías necesarias de DeepBeliefSDK, para esto debemos copiar los ficheros “libjpcnn.so” y “libjpcnn.h” en los directorios “/usr/lib/” y “/usr/include” respectivamente. Una vez hecho esto ejecutamos “make clean” para limpiar lo compilado previamente y “make GEMM=eigen TARGET=pi2” para compilar, estableciendo que usaremos la librería EIGEN y que se trata de una Raspberry Pi 2.

Compilado el programa de reconocimiento de imágenes, debemos activar el módulo de la pantalla TFT, previamente añadido el módulo al kernel, para ello ejecutamos el script tft-start que podemos encontrar en el Apéndice 1 – 7.3.

Establecidos los pasos previos, podremos usar el programa, cabe destacar que debemos hacerlo como superusuario, esto se debe al tratamiento de los pines GPIO, para poder comunicarnos con ellos debemos tener los permisos que este usuario nos concede.

3.9 Análisis

Para concluir el desarrollo, se realizaron una serie de mediciones de tiempo demostrando la viabilidad del proyecto, ya que de nada sirve el trabajo realizado si no se puede utilizar en la práctica.

Medimos el tiempo de reconocimiento de tres imágenes con una tabla comparativa entre la Raspberry Pi 2 y un ordenador con un procesador de 4 núcleos a 2.4 GHz y 4 GB de RAM.

Las imágenes las podemos encontrar en el repositorio del proyecto [5] en el directorio data.

	Ordenador 2400MHz	Raspberry Pi 2 700 MHz	Raspberry Pi 2 con Overclock 1000 MHz
Imagen 1 (dog)	236 ms	4585 ms	4462 ms
Imagen 2 (lena)	133 ms	4529 ms	3220 ms
Imagen 3 (pelota)	139 ms	4534 ms	3208 ms
Total	508 ms	13648 ms	10890 ms
Tiempo medio	169,3 ms	4549,3 ms	3630 ms

Tabla 3.1: Tiempos para reconocimiento

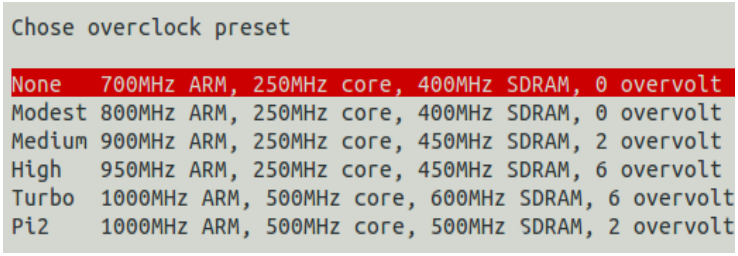

Figura 3.7: Overclock Raspberry Pi

Como podemos ver, la placa permite 6 modos de overclockeo, para esta tabla utilizamos el modo 1 (None) que viene por defecto y el modo 6 (Pi2). Analizando los resultados nos damos cuenta que frente a un ordenador convencional no tiene punto de comparación en lo que a tiempo respecta, sin embargo la movilidad que nos proporciona la Raspberry Pi no la proporciona un ordenador.

También podemos comprobar que con Overclock disminuimos el tiempo en casi 2 segundos, no obstante deberíamos contar con algún tipo de refrigeración para el procesador, de lo contrario la vida útil de nuestra placa sería bastante más corta.

Capítulo 4
Presupuesto

Para este proyecto hacen falta materiales hardware, que, a pesar de haber sido prestados por la universidad y el tutor del trabajo, tienen un costo económico. El software utilizado fue, o bien open source, por lo que no lleva costo económico alguno, o bien desarrollado por el alumno, al cual realizaremos una estimación por las 300 h. estipuladas en la Guía Docente de la asignatura.

Descripción	Costo
Raspberry Pi 2 Model B	40€
Pantalla TFT ili 9341	35€
Raspberry Pi Camera	30€
Horas de trabajo	300h * 10€ = 3000€
TOTAL	3105€

Tabla 4.1: Presupuesto

Capítulo 5
Conclusiones y líneas futuras

Podemos decir que es posible aplicar técnicas basadas en el Deep Learning en dispositivos computacionalmente limitados, teniendo como ventajas la movilidad, el bajo consumo y, dependiendo de las necesidades, podemos adaptarlo a diferentes usos.

A su vez, el Deep Learning se encuentra en continuo avance, por lo que el proceso de optimización para sus cálculos aún está en desarrollo.

También vemos cada vez más potencia en dispositivos más pequeños y económicos. Sin ir más lejos, ya está en el mercado la Raspberry Pi 3, que cuenta con un procesador de 64 bits y representa una potencia de unas diez veces la de la Raspberry Pi original o más que duplicar la potencia de una Raspberry Pi 2.

Por ello, podemos concluir que la línea de estudio de nuestro proyecto puede seguir en continuo desarrollo, pudiendo implementarse en dispositivos reducidos cada vez más potentes.

Como posible evolución del trabajo, podría pensarse en un detector de objetos peligrosos para personas ciegas. Con una red entrenada adecuadamente, en la que las imágenes puedan ser iconos de peligro u objetos punzantes, por ejemplo; añadiendo un dispositivo de salida de audio, como un altavoz, tendríamos un detector de peligro que ayude a personas con esta discapacidad en su vida cotidiana.

Relacionado con lo anterior, cabe la posibilidad de aumentar la capacidad de la aplicación introduciendo la característica de detectar la posición del objeto en la imagen en lugar de sólo detectar su presencia. Con ello se podría dar alguna ayuda para poder dirigirse hacia el objeto (o evitarlo).

Otra posible línea de desarrollo es alguna implementación de estas técnicas aprovechando la potencia de las GPUs de los Smartphones, a través de APIs de cálculo como RenderScript para dispositivos Android o CUDA en tablets basados en chipsets de NVIDIA.

Por otra parte, a nivel personal, la realización de este proyecto me a servido para introducirme en el mundo de las redes neuronales, ya que en el progreso de la carrera no se hace demasiado hincapié en este ámbito.

Además he descubierto numerosas aplicaciones y proyectos en curso relacionados con este tema, que me han llamado la atención y me motivan a seguir estudiando en esta línea de desarrollo.

Capítulo 6
Summary and Conclusions

We can say that it is possible to apply techniques based on Deep Learning in computationally limited devices, with the advantages of mobility, low consumption and, depending on the needs, we can adapt the application to several uses.

Deep Learning is in continuous progress, so the optimization process for its calculations is still a moving target.

We also see more and more power in smaller and cheaper devices, without going any further, the Raspberry Pi 3 is already on the market, which has a 64-bit processor that provides a performance of a factor of ten over the original Raspberry Pi or more than duplicate the performance of the Raspberry Pi 2.

So we can conclude that the line of our project will be in continuous development, making possible to build implementations in increasingly powerful devices.

As a possible evolution of this work, one could think of a detector of dangerous objects for blind people. With a network adequately trained, in which images can be icons of danger or sharp objects, for example; adding an audio output device such as a speaker, we would have a hazard detector that helps people with this disability in their daily lives.

Related to last point, there exists the possibility of augmenting the application introducing the ability of detecting a desired object position instead of just its presence. This would allow giving some help for moving towards it (or avoiding it)

Another possible line of development is some implementation of these techniques leveraging the power of GPUs smartphones, through calculation as RenderScript APIs for Android devices or CUDA in tablets based NVIDIA chipsets.

On the other hand, in personally level, the working on this project introduced me in the neuronals networks and the deep learning, inasmuch as in my university progress no too much emphasis on this area.

I also discovered many aplications and projects in progress related with this topic, that it motivate me to continuing studying in this line of development.

Capítulo 7
Apéndice 1: Algoritmos

7.1 Algoritmo interzaf.py

```
#!/usr/bin/python
#################################################################
###   Autor:       Rodrigo Colombo Vlaeminch                  ###
###   Universidad: Universidad de La Laguna                   ###
###   Descripcion: Script de enlace entre la libreria de      ###
###                reconocimiento de imagenes realizada       ###
###                por Pete Warden y la captura de imagen     ###
###                de la Raspberry Pi 2.                      ###
###   Version:     V3.1                                       ###
#################################################################

import picamera
import time
import sys
from subprocess import call
import RPi.GPIO as GPIO                          #import RPi.GPIO module
GPIO.setmode (GPIO.BCM)                          #choose BCM or BOARD
GPIO.setup (23, GPIO.IN, GPIO.PUD_UP)            #set GPIO23 as an input
GPIO.setup (18, GPIO.IN, GPIO.PUD_UP)            #set GPIO18 as an input
def envia_foto (foto):
  if len (foto) > 0:
    ejecutar = ['fbi','-d','/dev/fb1','-T','1','-noverbose','-a',foto]
    call (ejecutar)
  else:
    print "No hay foto"
def capturar ():
  camera = picamera.PiCamera ()
  camera.resolution = (320, 240)   #tamanio por defecto de la imagen
```

```
  name = str (camera.resolution)
  name = str
('/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/img'+name+'.jpg')
#asignacion del nombre de la imagen
  camera.capture (name)            #captura
  envia_foto (name)
  camera.close ()
  return name
def resultado (fichero):
  ejecutar = ['sort','-n','-k2','-r',fichero,'-
o','/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/log2.txt']
#ordenamos el resultado para que salga la mayor probabilidad primero
  call (ejecutar)
  res = open ('/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/log2.txt',
'r')                                        #abrimos fichero resultado
  linea = res.readline ()  #leemos la primera linea
  linea = linea.split ()   #separamos la linea en una lista de palabras
  if float (linea[0]) > 0.05:
    res = "Resultado:" + str (linea[1])
    envia_foto (res)            #enviamos el resultado a la pantalla
  else:
    envia_foto ("imposible reconocer")
  time.sleep (2.5)
def reconoce (foto):
  if len (foto) > 0:
    ejecutar = ['/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/jpcnn','-
i',foto,'-
n','/home/pi/rodrigo_proyecto/2016/Pi2Deep/networks/jetpac.ntwk','-t','-
m','s','-d']
    outfile = open
('/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/log.txt', 'w')
    errfile = open ('/dev/null', 'w')
    call (ejecutar,stdout = outfile,stderr = errfile)
#ejecucion del reconocimiento
    outfile.close ()
    errfile.close ()
    resultado ('/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/log.txt')
  else:
    print "error en reconocimiento"
def salir (foto):
  envia_foto ("Fin del programa")
  ejecutar = ['rm','-
r',foto,'/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/log.txt','/home/pi
/rodrigo_proyecto/2016/Pi2Deep/source/log2.txt']
```

```
  call (ejecutar)
  GPIO.cleanup ()               #reset de los puertos GPIO usados
  print "Bye bye"
fotos = {1:
'/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/f_menu/Capturar.jpg',
         2:
'/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/f_menu/Previsualizar.jpg',
         3:
'/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/f_menu/Reconocer.jpg',
         0:
'/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/f_menu/Salir.jpg',
         10: '/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/img(320,
240).jpg'
       }
def op1 ():
  envia_foto (capturar ())
  time.sleep (3)
def op2 ():
  envia_foto (fotos[10])
  time.sleep (4)
def op3 ():
  re = '/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/img(320, 240).jpg'
  reconoce (re)
  envia_foto (fotos[10])
def op0 ():
  salir (fotos[10])
  call ('poweroff')      #Apagamos la maquina
  exit (0)
opcion_menu = {1: op1,
               2: op2,
               3: op3,
               0: op0,
              }
contador_menu = 1
call ('/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/tft-start')
#inicializamos el modulo de la pantalla
envia_foto (fotos[1])
try:
  while True:
    if (GPIO.input (23) == 0):       #boton de desplazamiento de menu
      contador_menu = int ((contador_menu + 1) % 4)
      envia_foto (fotos[contador_menu])
      time.sleep (0.3)
```

```
    if (GPIO.input (18) == 0):                    #boton de accion
      opcion_menu[contador_menu] ()
      contador_menu = 1
      envia_foto (fotos[contador_menu])
      time.sleep (0.3)
except KeyboardInterrupt:
  salir (fotos[10])
```

7.2 Algoritmo demonio-interfaz

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:           demonio-interfaz
# Required-Start:      $remote_fs $syslog
# Required-Stop:       $remote_fs $syslog
# Default-Start:       2 3 4 5
# Default-Stop:        0 1 6
### END INIT INFO

PATH=/bin:/sbin:/usr/bin
DAEMON=/home/pi/rodrigo_proyecto/2016/Pi2Deep/source/interfaz.py
NAME=interfaz
DESC=interfaz
# Nombre de usuario para la ejecucion (Default: root).
USER=root
# pidfile
PIDFILE=/var/run/$NAME.pid
[ -x $DAEMON ] || exit 0
checkpid() {
   [ -f $PIDFILE ] || return 1
   pid=`cat $PIDFILE`
   [ -d /proc/$pid ] && return 0
   return 1
}
case "${1}" in
  start)
   echo -n "Starting ${DESC}: "
```

```
        start-stop-daemon --start --quiet --pidfile ${PIDFILE} \
        --chuid ${USER} --background --make-pidfile \
        --exec ${DAEMON}
    echo "${NAME}."
    ;;
  stop)
    echo -n "Stopping ${DESC}: "
    start-stop-daemon --stop --quiet --pidfile ${PIDFILE} \
    --oknodo
    echo "${NAME}."
    ;;
  restart|force-reload)
    echo -n "Restarting ${DESC}: "
    start-stop-daemon --stop --quiet --pidfile ${PIDFILE} \
    --oknodo
    sleep 1
    start-stop-daemon --start --quiet --pidfile ${PIDFILE} \
    --chuid ${USER} --background --make-pidfile \
    --exec ${DAEMON}
    echo "${NAME}."
    ;;
  *)
    N=/etc/init.d/${NAME}
    echo "Usage: ${NAME} {start|stop|restart|force-reload}" >&2
    exit 1
    ;;
  esac
  exit 0
```

7.3 Algoritmo tft-start

```
#!/bin/bash
`modprobe fbtft_device debug=7 custom name=fb_ili9341 speed=16000000 rotate=90 gpios=dc:25 init=-1,0x01,-2,5,-1,0x28,-1,0xEF,0x03,0x80,0x02,-1,0xCF,0x00,0xC1,0x30,-
```

1,0xED,0x64,0x03,0x12,0x81,-1,0xE8,0x85,0x00,0x78,-
1,0xCB,0x39,0x2C,0x00,0x34,0x02,-1,0xF7,0x20,-
1,0xEA,0x00,0x00,-1,0xC0,0x23,-1,0xC1,0x10,-1,0xC5,0x3e,0x28,-
1,0xC7,0x86,-1,0x3A,0x55,-1,0xB1,0x00,0x18,-
1,0xB6,0x08,0x82,0x27,-1,0xF2,0x00,-1,0x26,0x01,-
1,0xE0,0x0F,0x31,0x2B,0x0C,0x0E,0x08,0x4E,0xF1,0x37,0x07,0x1
0,0x03,0x0E,0x09,0x00,-
1,0xE1,0x00,0x0E,0x14,0x03,0x11,0x07,0x31,0xC1,0x48,0x08,0x0
F,0x0C,0x31,0x36,0x0F,-1,0x11,-2,100,-1,0x29,-2,20,-3`

Bibliografía

[1] http://www.cs.toronto.edu/~hinton/absps/imagenet.pdf

[2] https://github.com/jetpacapp/DeepBeliefSDK

[3] http://eigen.tuxfamily.org/index.php?title=Main_Page

[4] https://github.com/notro/fbtft

[5] https://github.com/rodrig92/Pi2Deep

[6] http://arxiv.org/pdf/1310.1531v1.pdf

[7] https://www.csie.ntu.edu.tw/~cjlin/libsvm/

[8] http://www.broadcom.com/docs/support/videocore/VideoCoreIV-AG100-R.pdf

[9] http://libccv.org/

[10] https://es.wikipedia.org/wiki/Aprendizaje_profundo

[11]http://www.arm.com/products/processors/technologies/neon.php

[12] https://gigaom.com/2013/11/01/the-gigaom-guide-to-deep-learning-whos-doing-it-and-why-it-matters/

[13] https://rubenlopezg.wordpress.com/2014/05/07/que-es-y-como-funciona-deep-learning/

[14] http://www.eldiario.es/hojaderouter/tecnologia/software/moda-deep_learning-algoritmo-inteligencia_artificial_0_275772610.html

[15] https://es.wikipedia.org/wiki/Raspberry_Pi

[16] http://www.xatakahome.com/trucos-y-bricolaje-smart/probamos-la-nueva-raspberry-pi-2-a-fondo