Universidad de Sevilla

Escuela Técnica Superior de Ingeniería

Sistema de Reconocimiento Facial basado en Redes Neuronales Convolucionales sobre el dispositivo Raspberry Pi

Máster en Ingeniería Electrónica, Robótica y Automática
Por Javier Méndez Gómez

1 de Agosto de 2019

Javier Méndez Gómez

SUMMARY

Keywords: Convolutional Neural Networks, CNN, Raspberry Pi, Python

Abstract

Training and implementation of Convolutional Neural Networks (CNN) using Python programming language and TensorFlow on a Raspberry Pi device for facial verification. During this project, different types of CNN structures will be tested in order to study the results obtained for each of them as well as their operation on the Raspberry Pi 3B device.

RESUMEN

Palabras clave: Redes neuronales convolucionales, CNN, Raspberry Pi, Python

Resumen

Entrenamiento e implementación de Redes Neuronales Convolucionales (CNN) utilizando el lenguaje Python y TensorFlow sobre un dispositivo Raspberry Pi para verificación facial. Durante este proyecto se probaran distintos tipos de estructuras de CNN con el objetivo de estudiar los resultados obtenidos para cada una de dichas estructuras así como su funcionamiento sobre el dispositivo Raspberry Pi 3B.

Agradecimientos

Gracias a todos aquellos que me han apoyado y me han impulsado a seguir aprendiendo durante estos años, ya que un buen ingeniero nunca deja de aprender.

Tabla de contenidos

CAPÍTULO I

MOTIVACIÓN Y OBJETIVO

Se ha decidido desarrollar este proyecto a la vista del reciente auge del Deep Learning en casi todos los campos, como puede ser el control de calidad, sector sanitario y reconocimiento de patrones entre otros. Los beneficios que las redes neuronales o el Deep Learning nos proporcionan son una mayor facilidad a la hora de su programación con respecto a técnicas tradicionales, donde el programador debe programar el proceso de clasificación/predicción, así como unos mejores resultados cuando se cumplen ciertos requisitos que se explicarán más adelante.

Una de las principales razones para el aumento del uso de las redes neuronales es el del Big Data, es decir, grandes cantidades de datos que antes no se podían procesar de forma sencilla, pero gracias a los avances tecnológicos actuales se pueden procesar mediante el uso de GPU y servidores online. Las redes neuronales permiten el uso de estos datos para la extracción de características y clasificación/previsión de datos, superando a las tecnologías previas.

En este proyecto se explicará cómo se ha desarrollado un sistema de verificación facial basado en el uso de dichas redes neuronales, en concreto redes neuronales convolucionales, y Raspberry Pi.

El objetivo principal del proyecto será la búsqueda de una estructura para la red neuronal que proporcione un buen resultado y, posteriormente al entrenamiento de dicha red, el uso de esta en el dispositivo Raspberry Pi.

La idea del proyecto es trasladar la inteligencia proporcionada por Deep learning a dispositivos pequeños y cercanos a la aplicación, como alternativa a un procesamiento en la nube o a la transmisión de datos a otros equipos donde se realiza el procesamiento de la información. Este enfoque es conocido como Edge Computing, que permite reducir los tiempos desde que se genera un dato, se procesa y se obtienen los resultados buscados.

Los objetivos particulares de este proyecto son:

1. Buscar una estructura adecuada para la CNN (Convolutional Neural Network) y base de datos.
2. Entrenar y mejorar la CNN.
3. Implementar la red neuronal en el dispositivo Raspberry Pi.
4. Identificar las limitaciones de los dispositivos Raspberry Pi para ejecutar redes neuronales.

CAPÍTULO II

ESTADO DEL ARTE

La inteligencia artificial se ha definido de diversas maneras a lo largo de la historia pero la definición generalmente aceptada es la capacidad de un ordenador o máquina de pensar o la capacidad de actuar inteligentemente. La primera de estas definiciones se centra en la idea de inteligencia humana, es decir, el proceso que se realiza para llegar a una conclusión, mientras que la segunda definición se centra en los resultados que se obtienen sin centrarse en el proceso seguido para obtener dichos resultados.

Una de las ramas principales de la inteligencia artificial hoy en día es el Machine Learning que busca crear programas con los que se permita que el propio ordenador aprenda a realizar una tarea de forma automática mediante un estudio previo de las características de los datos por el programador, ya que en esta técnica es el usuario el que indica que características se deben estudiar. Gran parte de la inteligencia depende del proceso de aprendizaje seguido por lo que esto es otro de los parámetros que el programador debe elegir con el fin de mejorar los resultados. Esta técnica se recomienda cuando no se tienen grandes cantidades de datos para entrenar la red, ya que al ser el programador el que elige las características a estudiar estamos simplificando el proceso de aprendizaje. En un principio el objetivo era la búsqueda y reconocimiento automático de patrones pero evolucionó mediante el uso del razonamiento probabilístico y estadística, a la vez que continuó profundizando en el reconocimiento de patrones.

Las redes neuronales forman parte del Machine Learning, para ser específicos están incluidos en el Deep Learning.

1 Deep learning

El Deep Learning es una rama del Machine Learning que surge como una evolución de este. Puede definirse como el uso de grandes cantidades da datos por parte de una IA (inteligencia artificial) durante un proceso de aprendizaje autónomo donde el propio dispositivo, mediante el proceso de entrenamiento, aprende a distinguir las características importantes a la hora de realizar la clasificación, por lo que el usuario no tiene que realizar un preprocesamiento de los datos tan exhaustivo.

En la metodología Deep Learning, se hace uso de estructuras lógicas similares a la organización del sistema nervioso central biológico, en las que podemos encontrar capas de proceso (las llamadas neuronas artificiales) que se especializan en diferentes tareas, como puede ser detectar determinadas características de los datos que se le proporcionan.

Deep Learning está empezando a usarse en una gran variedad de campos, como puede ser la visión artificial, donde se obtiene una mejora considerable en comparación con técnicas más tradicionales, como puede ser el uso de bibliotecas como OpenCV.

La diferencia entre Machine Learning y Deep Learning, pese a ser el segundo una rama del primero, está en los algoritmos que se utilizan para el proceso de clasificación/predicción. En el primero, es el programador el que establece ciertos criterios básicos en función de los que el programa aprenderá como ya hemos explicado anteriormente, mientras que en el proceso del Deep Learning el programador no especifica qué características se deben buscar en los datos, solo la estructura de la red que se quiere utilizar, así como una serie de opciones de la red como puede ser el learning rate o el optimizador. El Deep Learning busca el algoritmo con una mayor precisión mediante la imitación del funcionamiento de neuronas biológicas, forma iterativa. Otra de las diferencias es la cantidad de datos necesarios para el proceso de aprendizaje, siendo mucho mayor en el Deep Learning pero proporcionando mejores resultados que con el Machine Learning cuando se cumple esta condición. En el caso de no tener muchos datos para el proceso de entrenamiento, es recomendable usar Machine Learning, para aliviar la carga durante el proceso de entrenamiento ya que no debe buscar que características usar.

2 Redes neuronales

Las estructuras básica de redes neuronales artificiales se basan en el Perceptrón, diseñado por Rosenblatt en 1959. Esta neurona artificial o perceptrón puede recibir varias entradas y, mediante su suma ponderada y una función de activación posterior, nos proporciona una salida que queremos que se aproxime a la salida deseada.

$$y = f(\sum(x_n w_n + b))$$

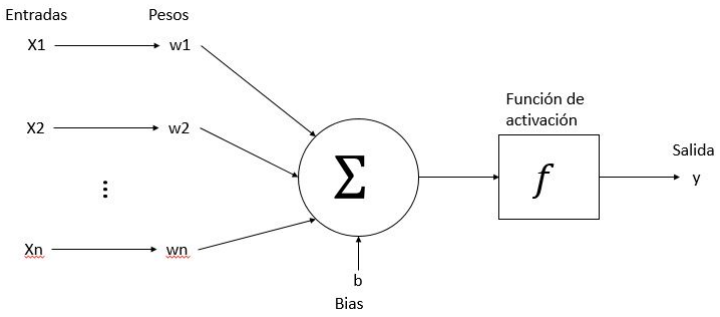

Figura II.1: Modelo de neurona artificial

La precisión del perceptrón es limitada ya que solo permite clasificar problemas linealmente separables, como indicaron Minsky y Papert.

Posteriormente, como una mejora del perceptrón, surgió el perceptrón multicapa, diseñador en 1986 por Rumelhart y otros autores. Esta estructura representa una mejora ya que permite la clasificación de problemas que no son linealmente separables a diferencia del perceptrón simple. Se basa en añadir capas intermedias entre la capa de entrada y la capa de salida a la estructura, denominadas capas ocultas con diferentes tipos de funciones de activación.

De esta forma, se diseña una red de varias capas con nodos interconectados capaces de procesar la información que se les proporciona y devolviendo un resultado a la capa siguiente, hasta llegar a la última capa que devuelve un resultado basado en todo este proceso. El resultado que se obtiene en cada capa depende de las denominadas funciones de activación.

Normalmente las redes de neuronas cuentan con con más de una neurona por capa y con varias capas como el caso del perceptrón multicapa antes mencionado. Estas neuronas pueden tener relación entre ellas o solo con las neuronas de capas próximas y anteriores, dependiendo del tipo de capa en el que nos encontremos. Cada una una de estas relaciones, es decir, los pesos entre las neuronas y los bias son los parámetros que se entrenan en la red.

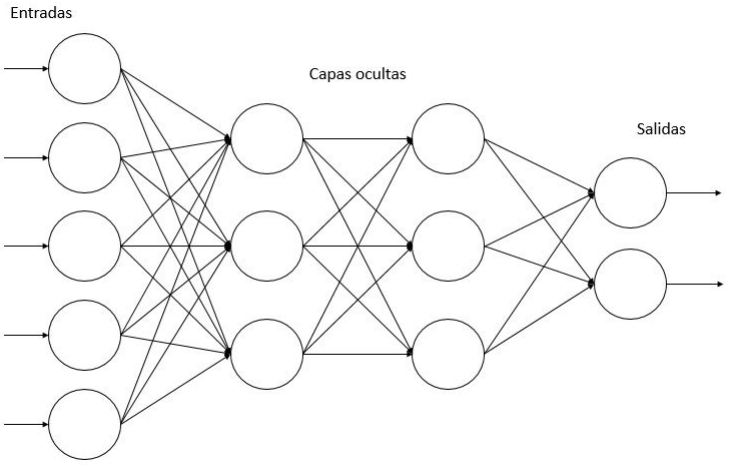

Figura II.2: Modelo con varias capas de neuronas artificiales

Durante el proceso de aprendizaje, los valores de los pesos y el bias son corregidos de forma iterativa con el objetivo de, tras suficientes iteraciones, conseguir que el resultado obtenido sea

el correcto para una tarea en concreto.

Uno de los procesos más populares para realizar esta operación es el llamado *backpropagation*. Este método consiste en calcular las salidas de cada capa hasta llegar a la capa final (etapa forward), donde se obtiene la salida. Este resultado obtenido se compara con el resultado deseado para calcular el error e intentar minimizarlo mediante el proceso de backpropagation en el que se busca el descenso en gradiente de la función de coste o error.

La función de activación también tiene un papel fundamental, por lo que hay varias funciones entre las que se pueden elegir para intentar mejorar nuestro modelo, como por ejemplo ReLu o Leaky ReLu.

3 Redes neuronales convolucionales

Las redes neuronales convolucionales son un tipo de red neuronal artificial multicapa feed-forward especialmente diseñada para trabajar con imágenes. Estas neuronas trabajan de forma similar a las neuronas reales que podemos encontrar en la corteza visual primaria de un cerebro biológico.

Las imágenes no se perciben de igual forma por un ordenador y por un humano, aunque podamos decir que este tipo de red neuronal funciona de forma similar, ya que el ordenador percibe las imágenes como conjuntos de matrices bidimensionales con valores relativos a la imagen en cada punto, es decir, píxeles.

Las redes neuronales convolucionales se componen de múltiples capas de filtros convolucionales y filtros de reducción de datos (*average pooling, max pooling, etc*). Después de cada capa convolucional, se añade una función para realizar un mapeo causal no-lineal. Hay distintas funciones, al igual que en las redes neuronales tradicionales, entre las que se debe elegir con el fin de obtener el mejor resultado posible.

Según se avanza en estas capas/filtros, se suele reducir el tamaño de las imágenes o matrices de salida de los filtros a la vez que se aumentan el número de capas, es decir, si partimos de una imagen RGB por normal general en cada filtro se irán reduciendo las dimensiones y aumentando el número de capas:

$$3(capasRGB)x150x150 \rightarrow 16x100x100 \rightarrow 32x85x85 \rightarrow ...$$

En el último paso en estas redes de neuronas artificiales, se encuentran neuronas de perceptrón o *fully connected* para realizar la clasificación final sobre las características extraídas mediante los filtrados previos.

Debido a la convoluciones que se realizan en estas redes neuronales convolucionales, este tipo de red neuronal es apta para poder clasificar todo tipo de datos donde estos estén distribuidos de forma continua en el mapa o imagen de entrada, y a su vez sean estadísticamente parecidos a lo largo de la imagen de entrada. Esto se debe a su invarianza a traslaciones dentro de la imagen, es decir, el resultado no se basa en la posición exacta de la característica que estudia, ya que subdivide la imagen en pequeños conjuntos que estudia de forma separada,

permitiendo así obtener el resultado deseado pese a posibles traslaciones de la región de interés dentro de la imagen, siempre que se haya entrenado adecuadamente. Por esta razón, son especialmente eficaces para clasificar imágenes así como tareas de percepción por computador.

3.1 Capa convolucional

En estas capas es donde se realiza la convolución del filtro con la imagen de entrada de ese filtro. La convolución consiste en una serie de multiplicaciones y sumas de los parámetros del filtro convolucional y los parámetros de las matrices bidimensionales o imágenes.

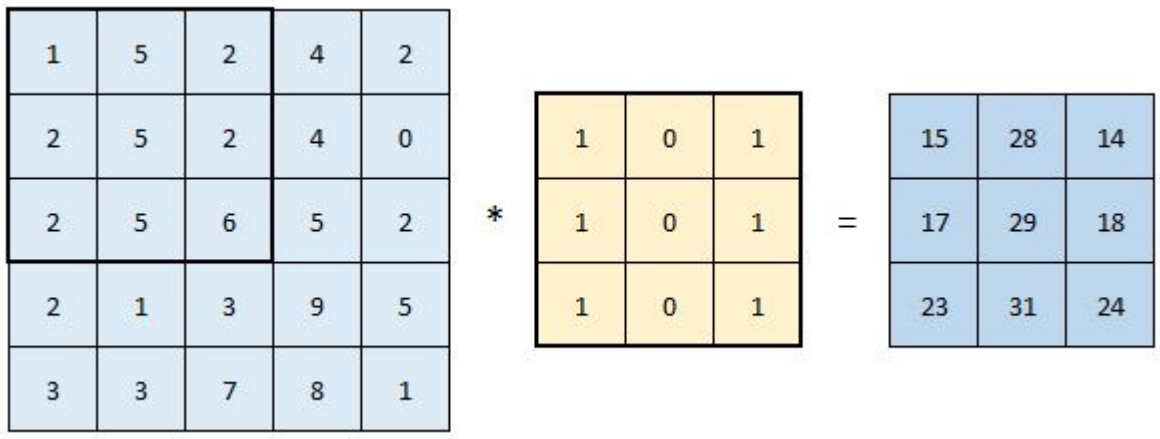

Figura II.3: Proceso de convolución

En este proceso se puede elegir si se quiere que la matriz resultante sea del mismo tamaño que la matriz de entrada o no, ya que el proceso de convolución reduce el tamaño. Para mantener el mismo tamaño se añaden filas y columnas auxiliares de valor cero. Esto no siempre es adecuado, ya que estamos evitando reducir el número de parámetros, pero a la vez tenemos más datos en los que intentar buscar patrones.

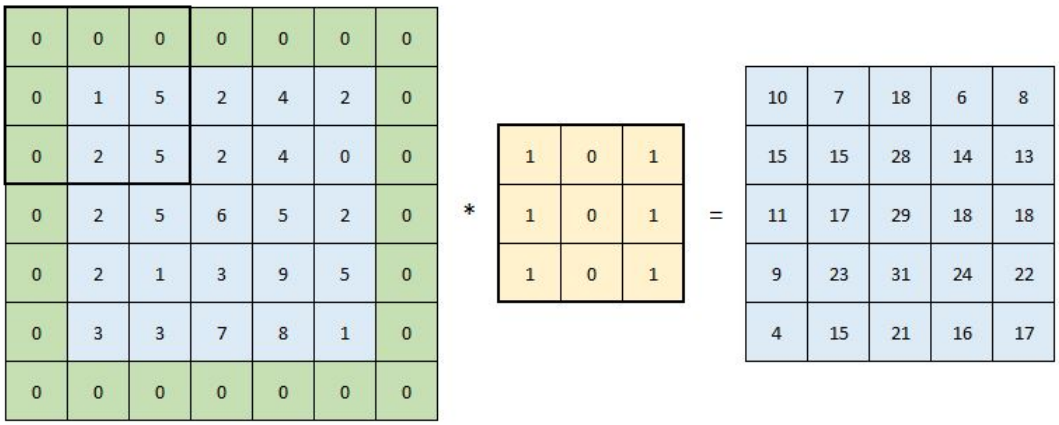

Figura II.4: Proceso de convolución con padding

Al final del proceso de aprendizaje, los parámetros o pesos del filtro se modifican con el fin de minimizar la función de coste o error obtenido, por lo que estos parámetros no con datos

que se puedan establecer a priori.

3.2 Capa pooling

En esta capa es donde se reduce el número de datos/tamaño de las matrices. Se suele colocar después de la capa convolucional. Estas capas son de gran utilidad ya que permiten reducir las dimensiones de la matriz bidimensional de la imagen que se estudia sin afectar al número de capas de dicha imagen. Esta operación también se llama reducción de muestreo debido a que la reducción de tamaño implica pérdida de información.

En la capa de pooling se puede elegir entre varias opciones:

1. *Average Pooling:* se calcula el valor medio de cada subconjunto de la matriz.
2. *Max Pooling:* se busca el valor máximo dentro de cada subconjunto de la matriz.

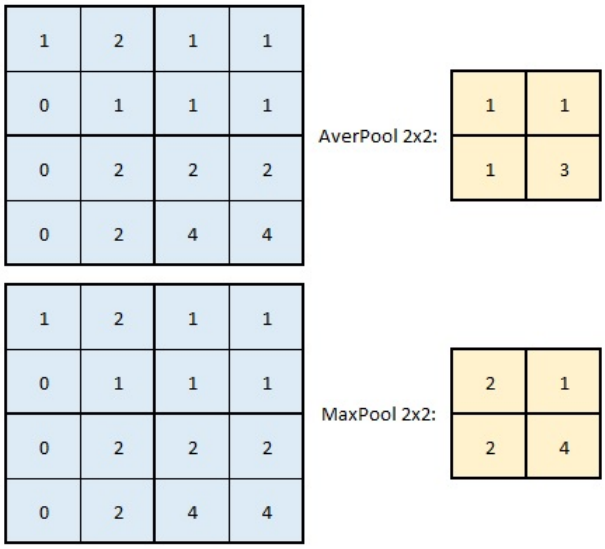

Figura II.5: AverPooling y MaxPooling 2x2

3.3 Capa totalmente conectada

Este tipo de capa suele encontrarse al final de la estructura de la red neuronal convolucional. Es esta capa todas las neuronas artificiales estas conectadas entre ellas, al igual que en las redes neuronales tradicionales como el perceptrón multicapa, con el fin de clasificar usando los datos o características obtenidos mediante las convoluciones y reducciones de parámetros.

Antes de esta capa, debe incluirse el proceso de *flattened*, que consiste en convertir la matriz bidimensional en un único vector de parámetros.

3.4 Capa dropout

La capa *dropout* es uno de los posibles métodos de regularización que ayudan a evitar el *overfitting*, es decir, evita que la red aprenda con gran precision a identificar las imágenes del conjunto de entrenamiento a costa de perder generalidad.

Esta capa suele usarse después de las capas totalmente conectadas, pero se pueden usar también después de las capas convolucionales o recurrentes.

La capa *dropout* elimina, de forma aleatoria y solo durante el proceso de entrenamiento de la red, algunas conexiones ocultas en la capa previa con una probabilidad variable configurada mediante el código del programa, siendo la probabilidad del 0.25-0.5% de forma general.

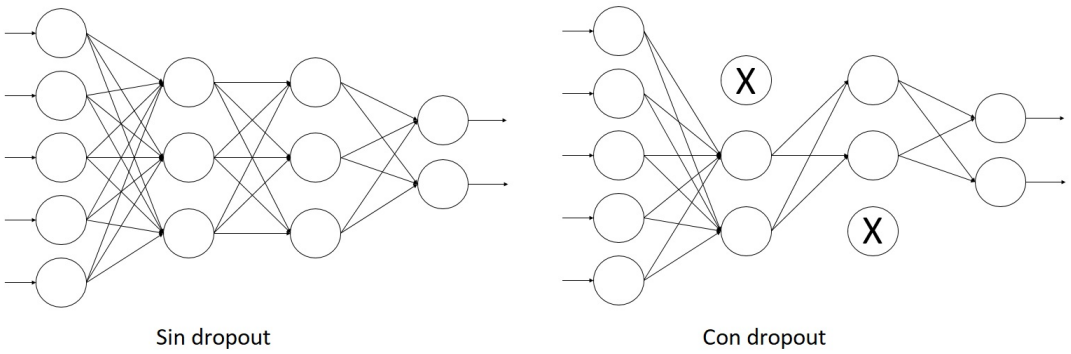

Figura II.6: Proceso dropout

4 Usos de las CNN

Actualmente las redes neuronales se están utilizando para una gran cantidad de tareas, esto es debido a que se ha comprobado que mediante el entrenamiento con una gran cantidad de datos nos proporcionan mejores resultados que métodos tradicionales de clasificación como puede ser un clasificador logístico o un árbol de decisión.

El inconveniente de las redes neuronales es la gran cantidad de datos que se necesitan durante el proceso de entrenamiento como se ha comentado, por lo que la capacidad de procesamiento también es muy elevada. Debido a esto, no ha sido hasta recientemente con el avance del Big Data que se ha comenzado a explotar el potencial de las redes neuronales.

Actualmente, las CNN se usan en muchos ámbitos, desde la medicina para el estudio de imágenes médicas [1] (resonancias, ecografias, etc) y detección de enfermedades, control de calidad hasta la conducción autónoma [2]. Ya que gran parte de los problemas se basan es detección de rasgos o características visuales que se pueden reconocer mediante el uso de las CNN pero que serían muy difíciles a simple vista para el ojo humano o que no se pueden realizar lo suficientemente rápido.

CAPÍTULO III

DESARROLLO DEL PROYECTO

Este proyecto se puede separar en varias fases:

1. Elecciones tecnológicas
2. Preprocesamiento del dataset
3. Elección y entrenamiento de la estructura de la CNN
4. Comparación de las estructuras

1 Elecciones tecnológicas

En este proyecto se busca estudiar el funcionamiento de redes neuronales sobre dispositivos de bajo coste por lo que lo primero es elegir el dispositivo que se va a usar. Las opciones más comunes son Arduino Y Raspberry Pi.

Para elegir entre Arduino Y Raspberry Pi se han comparado las siguientes características:

Características	Arduino uno	Raspberry Pi 3B
Programación	Arduino	Python
CPU	ATmega328P	Quad Core 1.2GHz Broadcom BCM2837 64bit
RAM	2 KB	1 GB
Precio	20€	31.5€

Tabla III.1: Comparación entre Arduino y Raspberry Pi

A la vista de esta tabla, se eligió utilizar la Raspberry Pi 3B debido, principalmente, a su mayor memoria RAM así como el lenguaje de programación que se utiliza con Raspberry Pi.

Una vez sabemos que dispositivo se usará, se va a explicar cómo se programarán las redes neuronales a lo largo de este proyecto.

Se usará Python, ya que es el lenguaje de programación por defecto que se usa para Machine Learning y Deep Learning, por lo que cuenta con bibliotecas específicas, como TensorFlow o Caffe, así como numerosos ejemplos donde se explica cómo usar dichas bibliotecas.

En concreto, usaremos las bibliotecas TensorFlow [3] y Keras [4] para redes neuronales junto con Python 3.6, ya que estas bibliotecas no tienen una version estable por el momento para el tradicional Python 2.7.

TensorFlow es una biblioteca desarrollada por Google para Deep Learning, específicamente trabaja con calculo numéricos basados en diagramas de flujo, y se puede usar en Python, JavaScript or Swift. Es de código abierto desde 2015, por lo que es una de las bibliotecas más populares actualmente.

TensorFlow tiene el inconveniente de que no es sencillo de aprender, ya que cuenta con un gran conjunto de instrucciones complejas, por lo que se suele usar junto con la biblioteca Keras, que permite un fácil uso de TensorFlow. Keras solo se puede utilizar en Python pero dado que es el lenguaje que se usará en este proyecto, no representa ningún inconveniente.

Keras fue integrado en la biblioteca TensorFlow in 2017, permitiendo un mayor nivel de abstracción a la hora de la programación.

2 Preprocesamiento del dataset

Para poder entrenar y comprobar el funcionamiento de la red neuronal se tiene que elegir un dataset específico para la aplicación, en este caso la verificación facial.

Existen numerosos dataset para esta aplicación como son 10k US Adult Faces Database [5], AT & T-The Database of Faces [6], Caltech Faces [7] o MIT-CBCL face recognition database [8]. Todos ellos nos proporcionan imágenes faciales de una gran cantidad de sujetos.

El dataset que se ha elegido para este proyecto es el dataset proporcionado, de manera libre, por el MIT para estudios faciales *"MIT-CBCL face recognition database"*. La elección se ha realizado en base al tamaño de las imágenes (con lo que intentamos que las imágenes sean similares a las que se pueden obtener con la Pi Camera de la raspberry Pi) así como al número de las imágenes de cada sujeto.

Este dataset nos proporciona 20 imágenes de 10 sujetos. Debido a las limitaciones a la hora del entrenamiento de la red neuronal, solo se han usado 4 sujetos. Lo que se estudiará a continuación sería escalable para un mayor número de sujetos.

Figura III.1: Sujetos estudiados en el dataset

Estos sujetos se han clasificado, de izquierda a derecha, como sujeto 1, sujeto 2, sujeto 3 y sujeto 4.

Como podemos ver, estas imágenes son todas del mismo tamaño, 152x152 píxeles, ya que el tamaño de imágenes que se usa para la red neuronal tiene que ser siempre el mismo.

En total se tienen 20 fotografías de cada sujeto, de las cuales se usarán 15 para el proceso de entrenamiento, 3 para validación y 2 para un test final con el que comprobar la precision de la red.

A la hora de saber cuántas imágenes son necesarias para el entrenamiento, existe una norma estadística:

$$Nparametrosdelared << Nimagenesdeentrenamiento$$

Por lo que es fácil intuir que necesitamos un mayor número de imágenes. Para ello, se usa el proceso de *data augmentation*, mediante el cual se generan imágenes artificiales usando las imágenes iniciales con el objetivo de aumentar el tamaño del dataset. Este proceso consiste en la rotación progresiva, desplazamiento del centro de la imagen y obtener la imagen especular de forma gradual entre otras técnicas para conseguir el número de imágenes necesarias.

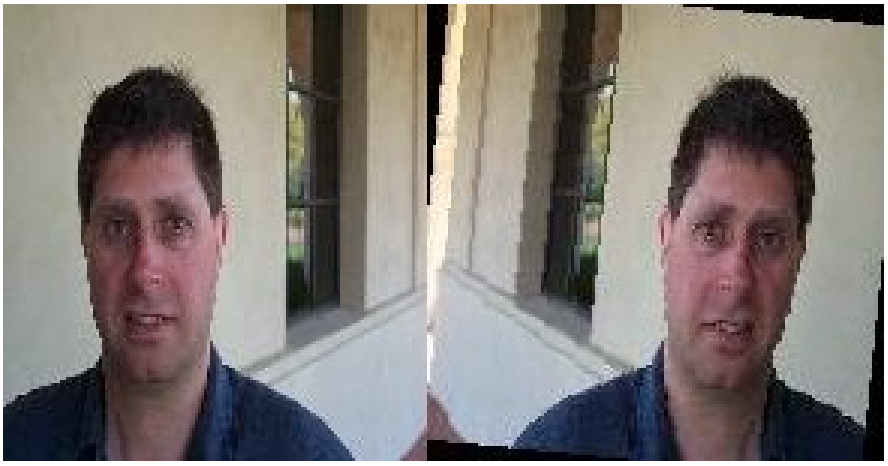

Figura III.2: Ejemplo data augmentation

Debido a que las imágenes en el dataset tienen distintos fondos, podemos suponer que la red neuronal no se basará en datos del fondo pero para evitar la posibilidad de que se base en el fondo, vamos modificar o preprocesar el dataset para que la red neuronal solo aprenda las caras. Esto lo vamos a realizar mediante la búsqueda y recorte de las caras de forma automática en el dataset antes de suministrar las imágenes a la red neuronal durante el proceso de entrenamiento.

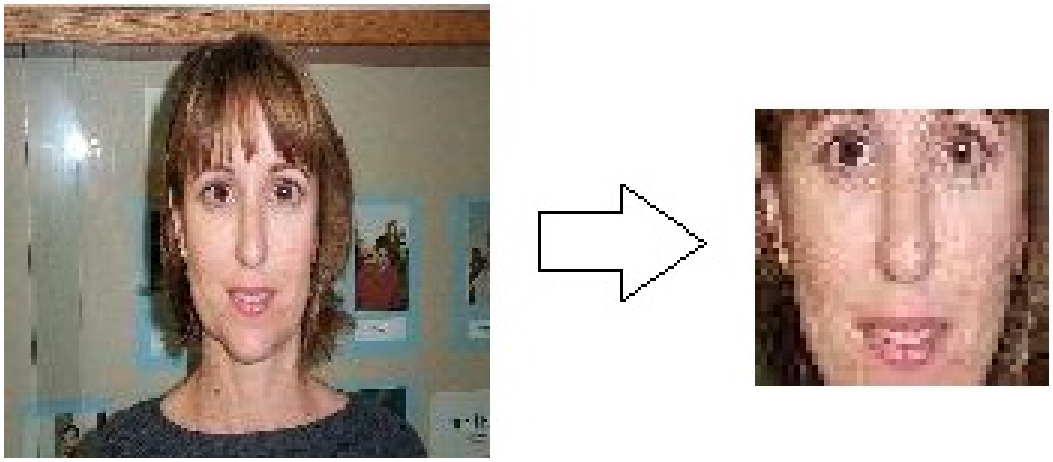

Figura III.3: Recortar caras

Al mismo tiempo, también se ha realizado otro preprocesamiento tradicional en este tipo de aplicaciones que es la normalización de las imágenes.

De igual forma, una vez que la red esté entrenada, habrá que aplicar los mismo preprocesamientos cuando se quiera ejecutar la red neuronal para clasificar una imagen.

3 Elección y entrenamiento de la estructura de la CNN

Una vez tenemos suficientes imágenes, se ha pasado a entrenar diferentes redes neuronales convolucionales con el objetivo de encontrar la mejor estructura para nuestro caso.

En este punto tenemos dos opciones, usar una red previamente entrenada y solo entrenar las ultimas capas o entrenar una red desde cero. La primera opción es más rápida y sencilla de ejecutar, ya que no necesitamos calcular todos los parámetros de la red. El inconveniente de este tipo de estructuras es que la precisión puede no ser tan buena como en una red entrenada desde cero en caso de que la aplicación para la que se entrenaron inicialmente no sea similar. En caso de aplicaciones similares, la precisión puede llegar a ser incluso mejor que en una red entrenada desde cero, ya que partimos de una estructura que se ha comprobado que da buenos resultados. Por este motivo, vamos a probar con dos redes ya entrenadas usando la técnica de "Transfer Learning".

La primera red que vamos a usar es la ResNet50 [9] donde los parámetros ya están entrenados y solo se entrenan lo de las capas posteriores que se añadirán. Las capas que se han añadido, las cuales si se entrenarán, son dos capas totalmente conectadas, la primera de 512 neuronas y la segunda de 256. La última capa que se ha añadido es una capa totalmente conectada de 4 neuronas, ya que esta capa se encargará de la clasificación final.

La segunda red que vamos a usar es la VGGnet16 [10]. En esta estructura, al igual que antes, las primeras capas están ya previamente entrenadas y solo se han añadido las mismas capas finales que en la ResNet50 para la clasificación final.

Al usar redes ya entrenadas, la carga computacional es mucho menor que en el caso de tener que entrenar una red neuronal desde cero. Gracias a esto, se ha podido realizar el proceso del entrenamiento en un ordenador con 6 GB de memoria RAM y 2GB de GPU y aun así el tiempo de entrenamiento ha sido de 45 minutos en ambos casos.

A continuación, se puede observar la precision obtenida con estas dos estructuras así como el número de parámetros de estas redes neuronales.

Estructura	Número de parámetros	Precisión
ResNet50	75.233.156	92.36
VGGnet16	24.417.092	84.72

Tabla III.2: Estructuras pre-entrenadas

Como se puede ver, el número de parámetros de la primera red, ResNet50, es elevado por que se comprobará más adelante la viabilidad de esta red sobre dispositivos de capacidades limitadas, como es el caso de una Raspberry Pi. Con respecto a la VGGnet16 previamente entrenada, podemos ver como se han obtenido buenos resultados y el número de parametros es inferior a la ResNet50 pero igualmente se debe estudiar su viabilidad más adelante.

Una vez que hemos estudiado algunas redes previamente entrenadas, se ha elegido entrenar algunas redes desde cero, con el objetivo de intentar buscar redes con mayor precisión y/o menor número de parámetros que las dos redes anteriores con el fin de facilitar su implementación es dispositivos de bajas prestaciones.

El proceso de entrenamiento de una red neuronal requiere una gran potencia de cálculo por lo que se suele utilizar la GPU con el fin de reducir los tiempos. Este proceso se puede realizar en el ordenador pero existen algunas otras opciones como es el uso de plataformas en la nube diseñadas para el proceso de entrenamiento de redes neuronales.

Una de estas plataformas es *Google Colab*, una plataforma de Google, similar a Google Drive, diseñada para ejecutar códigos que necesitan una memoria RAM superior a la que se tiene en el propio PC o una GPU de mejores características. *Google Colab* nos permite usar 13GB de memoria RAM del servidor de Google así como nos da acceso a la GPU Tesla K80, lo que permite reducir los tiempos de ejecución a una décima parte, de acuerdo con los datos proporcionados por Google [11].

En concreto se han probado seis estructuras para ser precisos. La primera estructura es la que se explica en el artículo de Taigman [12], una modificacion de dicha red que hemos llamado Taigman V2 a la que hemos añadidos más capas, la estructura LeNet5 [13], VGGNet 16 [14] (igual que la red pre-entrenada que se ha estudiando anteriormente pero sin guardar los pesos), ResNet20 [15] y una variante de esta última, que consiste en añadir una capa convolucional de 64 filtro más.

Taigman	Taigman V2	ResNet20	ResNet20 aumentada
Conv 32x11x11	Conv 32x11x11	Conv 32x3x3	Conv 32x3x3
MaxPool 3x3	MaxPool 3x3	Conv 32x3x3	Conv 32x3x3
Conv 16x9x9	Conv 16x9x9	MaxPool 2x2	MaxPool 2x2
LocallyCon 16x9x9	Conv 16x9x9	Conv 64x3x3	Conv 64x3x3
LocallyCon 16x7x7	LocallyCon 16x9x9	Conv 64x3x3	Conv 64x3x3
LocallyCon 16x5x5	Dense 4096	MaxPool 2x2	Conv 64x3x3
Dense 4096	Dense 4	Dense 512	MaxPool 2x2
Dense 4		Dense 128	Dense 512
		Dense 4	Dense 128
			Dense 4

Tabla III.3: Estructuras para CNNs (1)

LeNet5	VGGNet16
Conv 6x5x5	Conv 64x3x3
AverPool 2x2	Conv 64x3x3
Conv 16x5x5	MaxPool 2x2
AverPool 2x2	Conv 128x3x3
Conv 120x5x5	Conv 128x3x3
AverPool 2x2	MaxPool 2x2
Dense 84	Conv 256x3x3
Dense 4	Conv 256x3x3
	MaxPool 2x2
	Conv 512x3x3
	Conv 512x3x3
	MaxPool 2x2
	Conv 512x3x3
	Conv 512x3x3
	MaxPool 2x2
	Dense 4096
	Dense 4096
	Dense 4

Tabla III.4: Estructuras para CNNs (2)

A continuación, podemos ver la evolución de la precisión de cada una de estas estructuras. Todas ellas se han entrenado usando el mismo número de iteraciones así como el mismo conjunto de datos de entrenamiento y test, con el fin de poder comparar los resultados de todas estas estructuras.

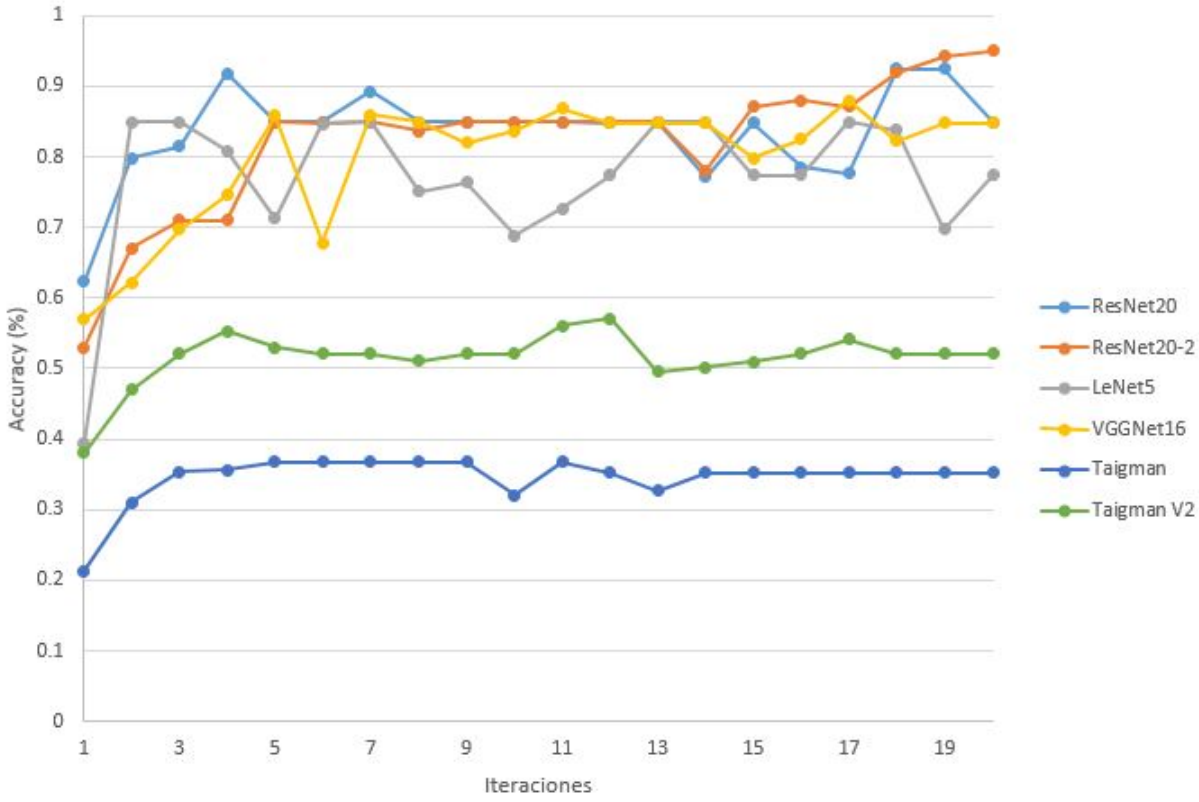

Figura III.4: Evolución de la precision en conjunto de test

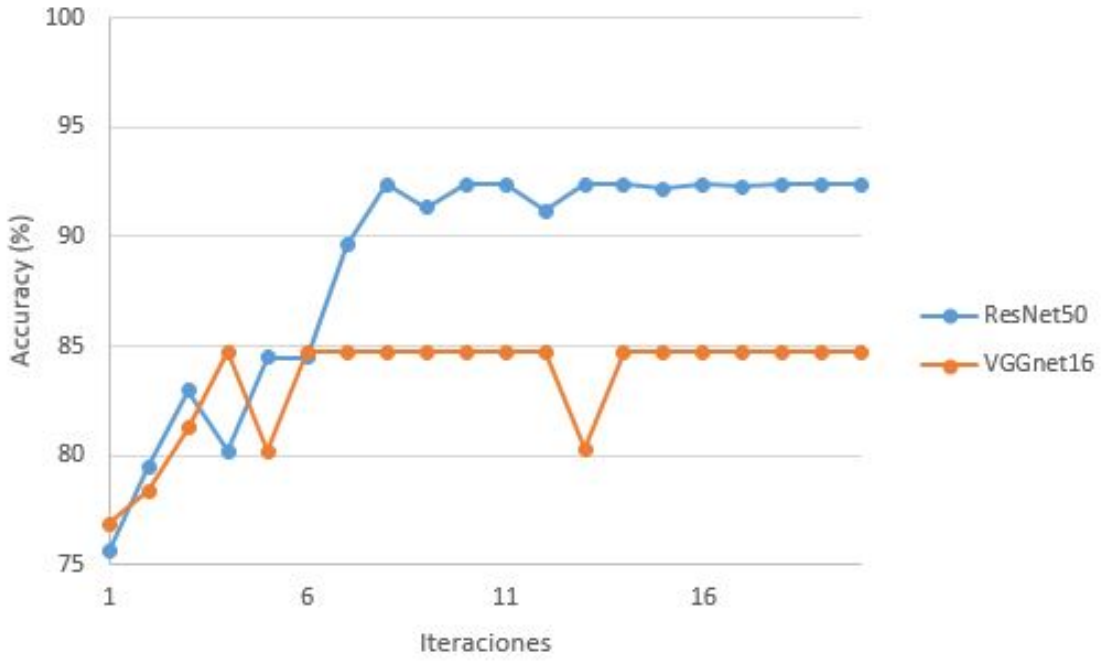

Figura III.5: Evolución de la precision en conjunto de test (redes pre-entrenadas)

En estas gráficas podemos comprobar como el número óptimo de iteraciones no es independientemente de la estructura, como ya sabíamos, ya que son completamente distintas (el número de parámetros, número de capas, dimensiones de cada capa, etc). Por ejemplo, pode-

mos ver que para el caso de las dos estructuras de Taigman, los resultados son muy estables desde las primeras iteraciones por lo que el número de iteraciones no afecta en gran medida, mientras que para otras estructuras la precisión varia de forma considerable de una iteración a otra, como es el caso de la estructura LeNet5, en la que hay variaciones de un 13% entre la cuarta y la quinta iteración.

Con respecto al overfitting, es decir, el caso en que la red neuronal aprende con gran precisión a reconocer las imágenes del conjunto de entrenamiento, pero los fallos con el resto de imágenes crece en gran medida, pasa lo mismo que con la precisión antes estudiada.

4 Comparación de las estructuras

Los resultados obtenidos para las estructuras antes mencionadas (eligiendo para cada estructura el número de iteraciones óptimas, siendo el mayor número posible 30 iteraciones) se pueden ver en la tabla de resultados, así como el número de parámetros de cada red. Estos resultados se han obtenido entrenando con 500 millones de imágenes mediante el uso de data augmentation, para cumplir la relación entre el número de parámetros dela red e imágenes de entrenamiento. La precisión se ha calculado como el porcentaje de aciertos en el conjunto de validación a lo largo del entrenamiento de la red.

Nombre de la estructura	Precisión en conjunto de test	Número de parámetros
Taigman	54.68%	569,942,340
Taigman V2	58.71%	300,433,954
ResNet20	85.37%	3,986,588
ResNet20 v2	94.51%	4,309,988
LeNet5	77.35%	3,203,688
VGGNet16	84.82%	59,459,646
ResNet50 pre-entrenada	92.36	75.233.156
VGGnet16 pre-entrenada	84.72	24.417.092

Tabla III.5: Resultados con diversas CNNs

A la vista de estas redes, se puede apreciar como las redes pre-entrenadas que se han estudiado tienen una precisión alta en comparación con las redes que se han entrenado desde cero, pero también tienen más parámetros en general.

Con respecto a las redes que se han entrenado desde cero, podemos ver como para las dos primera, los resultados obtenidos han sido muy inferiores a las otras estructuras que se han estudiado en este proyecto. Esto puede deberse a que se necesitan más imágenes para su entrenamiento. Pese a esto, debido a que partimos de 15 imágenes del conjunto de entrenamiento, no se han aumentado el número de imágenes ya que la calidad del dataset disminuye conforme aumentamos el número. Debido a esto, aunque aumentásemos el tamaño del dataset, los resultados no variarían.

Usando los criterios del menor número de parámetros posible con una alta precisión, se eligió la estructura ResNet20 modificada ya es que la que nos proporciona unos mejores resultados con nuestro dataset (94.51%) con un número de parámetros no excesivamente alto.

	Sujeto 1	Sujeto 2	Sujeto 3	Sujeto 4
Sujeto 1	2500	0	0	0
Sujeto 2	0	2500	0	0
Sujeto 3	0	0	2500	0
Sujeto 4	0	433	0	2067

Figura III.6: Matriz de confusion, ResNet20 V2

En esta tabla podemos ver los resultados obtenidos con la estructura ResNet20 modificada desglosados, con el fin de ver si las clases se clasifican igualmente bien o si hay un error asimétrico entre las clases.

Se puede apreciar como en este caso el error no es el mismo para todas las clases, ya que a los sujetos 1, 2 y 3 los clasifica correctamente en el 100% de los casos estudiados en el conjunto de test, mientras que para el sujeto 4 falla en el 17.32% de los casos.

También se ha comprobado cómo afecta la variación del número de sujetos a estudiar mediante la red neuronal una vez decidido cuál es la estructura óptima para esta aplicación entre todas las estructuras probadas. En concreto, se ha estudiado cómo afecta reducir el número de sujetos siendo 3 en vez de 4.

Al reducir el número de sujetos la precisión ha subido levemente ya que ahora la precisión es del 95.23%, como podíamos esperar ya que al haber menos clases el clasificador tiene que hacer un menor esfuerzo para diferenciarlas. Con respecto al tiempo de entrenamiento, no se ha notada una gran diferencia ya que la estructura es la misma y pese a ser un sujeto menos se han aumentado el número de imágenes de entrenamiento de las otras clases para poder seguir entrenando correctamente, ya que si reducimos el número total de imágenes del conjunto de entrenamiento ya no podríamos comparar los resultados.

En el caso de aumentar el número de sujetos a 5, la precision se ha mantenido como en el primer caso, ya que se ha obtenido una precisión del 94.37% y el tiempo de entrenamiento no ha variado es gran medida.

CAPÍTULO IV

ESTUDIO EN RASPBERRY PI

Una vez se ha probado el correcto funcionamiento en un ordenador se ha pasado al estudio sobre un dispositivo de bajo coste, Raspberry Pi 3B.

El objetivo es comprobar si se pueden obtener los mismo resultados mediante el uso de una cámara (PiCamera) en vez de suministrando imágenes previamente tomadas a la red neuronal.

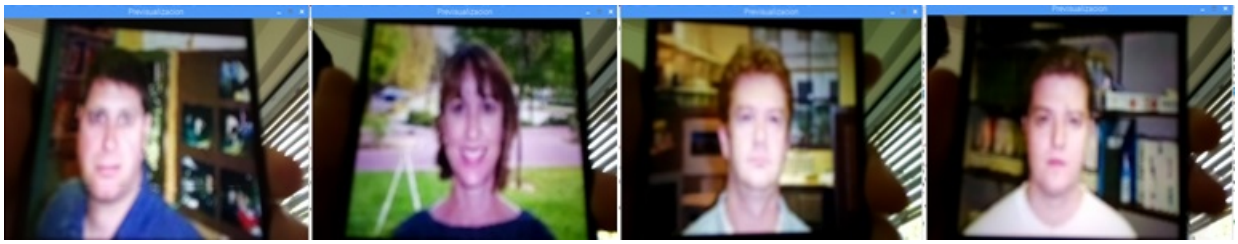

Figura IV.1: Imágenes de la cámara

Tras una prueba se comprobó que el código sigue funcionando igual que en el ordenador por lo que se pasó al estudio del tiempo.

Estudiaremos sobre el dispositivo los tiempos de ejecución de las estructuras para la CNN que se han estudiado, con el fin de poder tener en cuenta dichos resultados a la hora de elegir la red neuronal óptima para este proyecto.

De esta forma, podemos ver como los tiempos de ejecución de las diferentes redes neuronales estudiadas en este proyecto. En el caso de las dos primeras, al ser demasiado grandes, el dispositivo Raspberry Pi 3 no es capaz de ejecutarlas ya que no cuenta con suficiente memoria RAM para almacenar todos los parámetros. Por este motivo, estas dos estructuras se

Nombre de la estructura	Tiempo de ejecución (s)
Taigman	-
Taigman V2	-
ResNet20	2.05
ResNet20 v2	2.32
LeNet5	1.75
VGGNet16	11.34
ResNet50 pre-entrenada	15.63
VGGNet16 pre-entrenada	5.42

Tabla IV.1: Tiempos de ejecución

pueden descartar para su uso independientemente de la precisión.

También se quiere estudiar las limitaciones de las Raspberry Pi 3B utilizada con respecto al tamaño máximo de la red neuronal que puede ejecutarse antes de que los tiempos de ejecución sobrepasen un tiempo límite. Queremos estudiar esto ya que, si se tarda mucho en ejecutar el código, el programa no podría usarse para aplicaciones en tiempo real.

Vamos a ver la tabla de datos obtenidos donde podemos comparar el número de parámetros de una red con el tiempo de ejecución, creando para ello variantes de las estructuras usadas con un mayor/menor número de parámetros.

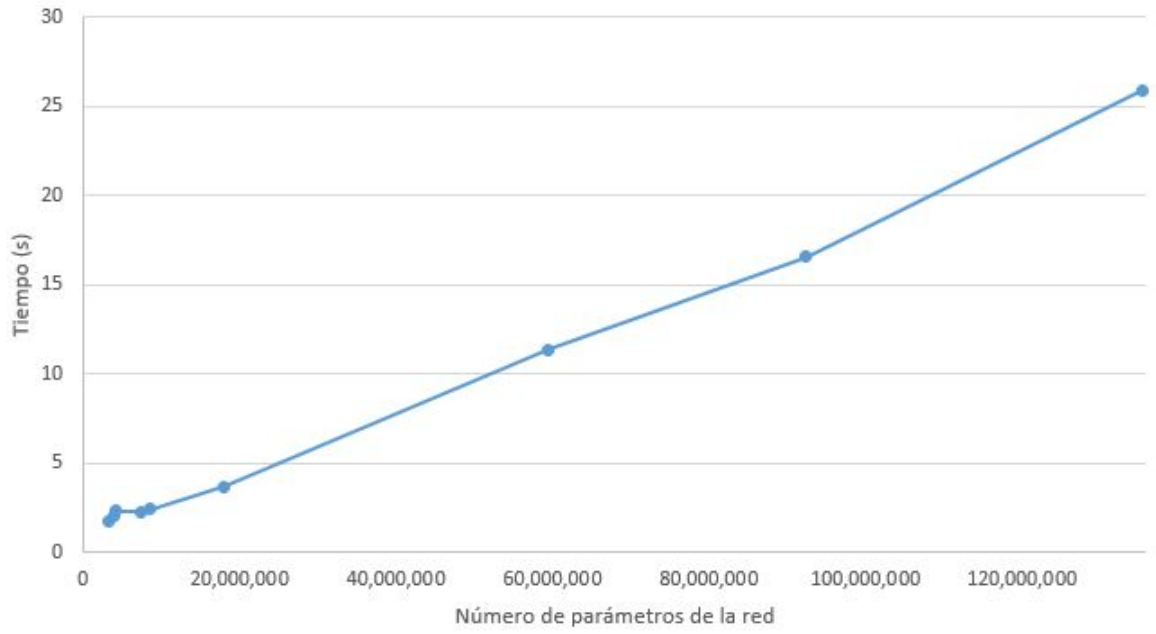

Figura IV.2: Tiempo de ejecucion de la CNN en el dispositivo RP

Al final, se ha podido comprobar que cuando el número de parámetros supera los 140 millones de parámetros, el dispositivo Raspberry Pi 3 no es capaz de ejecutar la CNN debido a las limitaciones de la memoria RAM. Para redes neuronales con numero de parámetros entre

50 millones y los 140 millones, el dispositivo es capaz de ejecutar la red neuronal pero los tiempos de ejecución son muy elevados y el dispositivo se calienta bastante.

Si lo comparamos con los tiempos de ejecución en la plataforma Google Colab, donde se tiene una RAM mucho mayor y una GPU dedicada, podemos ver cómo afecta la elección del dispositivo a los tiempos de ejecución.

Numero de parámetros	Tiempo de ejecución RP (s)	Tiempo de ejecución Google Colab (s)
3,203,688	1.757538462	0.00714
3,986,588	2.0578	0.00836
4,309,988	2.32568	0.009425
7,322,428	2.39821	0.009762
8,620,084	2.412307	0.0098438
18,028,426	3.6923	0.015102
59,459,646	11.33856	0.46062
92,412,848	16.58463	0.067356
135,489,204	25.89461	0.105178
176,548,254	-	0.165238
193,273,432	-	0.171524
206,626,048	-	0.183541
300,433,954	-	0.304525
569,942,340	-	0.556210

Tabla IV.2: Tiempos de ejecución en Raspberry Pi 3 y Google Colab

Podemos ver como los tiempos son mucho menores en la plataforma Google Colab, con diferencias de tres órdenes de magnitud. Por lo que queda claro que para un uso óptimo de las redes neuronales no solo hay que buscar la estructura adecuada, también tenemos que tener en cuenta las limitaciones del dispositivo físico sobre el que vamos a ejecutar dicha estructura.

CAPÍTULO V

CONCLUSIONES

A lo largo de este proyecto se ha podido comprobar la utilidad de las redes neuronales convolucionales para la verificación facial. Se han comparado distintas estructuras con el fin de intentar encontrar la más adecuada para este caso, pero pese a haberse encontrado una estructura que nos proporciona muy buenos resultados, nunca podemos afirmar que sea la óptima ya que hay un número infinito de posibles estructuras.

También se ha podido comprobar como el uso de redes previamente entrenadas, usando la técnica del Transfer Learning, pueden ser útiles para la verificación facial. Se ha podido ver que en algunos casos dan mejores resultados que redes neuronales entrenadas desde cero.

Al comparar los resultados obtenidos en un ordenador y un dispositivo de bajo coste, como es el caso de la Raspberry Pi, se ha comprobado como la elección de la estructura no debe realizarse sólo en función de la precisión que nos proporciona, si no también estudiando su aplicación sobre el dispositivo, ya que puede que los tiempos de ejecución sean demasiado grandes o que, debido a los recursos del propio dispositivo, sea irrealizable dicha estructura.

Pese a esto, se han conseguido unos resultados satisfactorios ya que la precisión alcanzada con la Raspberry Pi 3B ha sido del 94.51% con un tiempo de ejecución de 2.32 segundos, lo que puede que no permita su uso en tiempo real, pero si en aplicaciones que no requieran un tiempo de ejecución muy pequeño.

Se han estudiado dos opciones cuando se trabaja con redes neuronales, usar redes pre-entrenadas o entrar una red neuronal desde cero. Se ha visto como, haciéndose de forma adecuada, las dos opciones proporcionan buenos resultados. Las dos redes pre-entrenadas consiguieron resultados por encima del 80%, rivalizando con el resto de estructuras que se han entrenado desde cero con nuestra base de datos. Esto demuestra la eficacia del transfer learning pese a que en este caso se haya optado por usar una red entrenada desde cero.

En concreto, se han conseguido todos los objetivos principales de este proyecto, que eran:

1. Buscar una estructura adecuada para la CNN y base de datos.
2. Entrenar y mejorar la CNN.
3. Implementar la red neuronal en el dispositivo Raspberry Pi.
4. Identificar las limitaciones de los dispositivos Raspberry Pi para ejecutar redes neuronales.

CAPÍTULO VI

FUTURAS MEJORAS

Como mejoras futuras, sería interesante volver a repetir estas pruebas con el uso de la Raspbery Pi y un stick para aumentar sus capacidades, como puede ser el Movidius NCS, con el objetivo de comprobar los límites con dicho periférico, ya que se espera que los tiempos de ejecución sean menores e incluso puede que se puedan usar estructuras con un mayor número de parámetros.

Una opción para mejorar los resultados obtenidos en el dispositivo Raspberry Pi es el uso de ”sticks” o periféricos diseñados específicamente para mejorar el rendimiento del Deep Learning en dispositivos como la Raspberry Pi con la que se está trabajando.

Figura VI.1: Movidius NCS

Uno de estos periféricos es el Movidius NCS, el cual añade una VPU de alto rendimiento, lo que permite acelerar el procesamiento así como aumentar la memoria con lo que se puede trabajar con redes neuronales con un mayor número de parámetros. Este stick está diseñado para mejorar el rendimiento cuando se trabaja con Deep Learning y procesamiento de imágenes, por lo que permite evitar trabajar en la nube, acercándonos al Edge Processing.

Otra mejorar que se podría añadir es el estudio de los propios hiperparámetros de la red de forma iterativa, es decir, estudiar durante el proceso de entrenamiento si la estructura (número de capas y neuronas) es la óptima.

También podría diseñarse una carcasa e imprimirla con una impresora 3D para el dispositivo Raspberry Pi junto con la cámara y una pantalla, con el fin de tener un producto acabado con una interfaz sencilla para los usuarios.

APÉNDICES

A Código CNN ResNet20-V2

```
import numpy as np
import itertools
import keras
from keras import backend as K
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.convolutional import *
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
from keras.utils import to_categorical
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers import Conv2D, MaxPooling2D, LocallyConnected2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from PIL import Image, ImageOps

#Obtencion del dataset
train_path = 'Dataset/train'
valid_path = 'Dataset/valid'
test_path = 'Dataset/test'

train_batches = ImageDataGenerator().flow_from_directory(train_path,
    ↪ target_size=(152,152), classes=['1', '2','3','4'], batch_size=500000000)
test_batches = ImageDataGenerator().flow_from_directory(test_path,
    ↪ target_size=(152,152), classes=['1', '2','3','4'], batch_size=10000)
valid_batches = ImageDataGenerator().flow_from_directory(valid_path,
    ↪ target_size=(152,152), classes=['1', '2','3','4'], batch_size=10730000)

imgsTrain, labelsTrain = next(train_batches)
imgsTest, labelsTest = next(test_batches)
imgsValid, labelsValid = next(valid_batches)

print('Training data shape : ', imgsTrain.shape, labelsTrain.shape)
```

```
print('Testing data shape : ', imgsTest.shape, labelsTest.shape)
print('Valid data shape : ', imgsValid.shape, labelsValid.shape)

batch_size = 16
epochs = 30
num_classes = len(labelsTrain[1])

#Normalizacion
x_train = imgsTrain.astype('float32')
x_test = imgsTest.astype('float32')
x_valid = imgsValid.astype('float32')
x_train /= 255
x_test /= 255
x_valid /= 255

#Estructura de la CNN deeper ResNet20-V2
image_model = Sequential()
image_model.add(Conv2D(32, (3, 3), padding='same', input_shape=(152,152,3)))
image_model.add(Activation('relu'))
image_model.add(Conv2D(32, (3, 3)))
image_model.add(Activation('relu'))
image_model.add(MaxPooling2D(pool_size=(2, 2)))
image_model.add(Dropout(0.25))

image_model.add(Conv2D(64, (3, 3), padding='same'))
image_model.add(Activation('relu'))
image_model.add(Conv2D(64, (3, 3)))
image_model.add(Activation('relu'))
image_model.add(Conv2D(64, (3, 3)))
image_model.add(Activation('relu'))
image_model.add(MaxPooling2D(pool_size=(2, 2)))
image_model.add(Dropout(0.25))

image_model.add(Flatten())
image_model.add(Dense(128))
image_model.add(Activation('relu'))
image_model.add(Dropout(0.5))
image_model.add(Dense(num_classes))
image_model.add(Activation('softmax'))

opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)
image_model.compile(loss=keras.losses.categorical_crossentropy,
    ↪ optimizer=opt,metrics=['accuracy'])
image_model.summary()

#ENTRENAMIENTO
image_train = image_model.fit(x_train, labelsTrain,
    ↪ batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_valid,
    ↪ labelsValid))

#TEST
test_eval = image_model.evaluate(x_test, labelsTest, verbose=1)
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

Código fuente 1.1: CNN en Python

B Código detección y clasificación facial

```
import numpy as np
import itertools
import keras
from keras import backend as K
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
```

```
from keras.layers.convolutional import *
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
from keras.utils import to_categorical
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers import Conv2D, MaxPooling2D, LocallyConnected2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from PIL import Image, ImageOps

#Librerias para recortar imagen
import dlib
from skimage import io

#Libreria para test
from keras.preprocessing import image as im

#Definimos la estructura de la CNN
Newmodel = Sequential()
Newmodel.add(Conv2D(32, (3, 3), padding='same', input_shape=(152,152,3)))
Newmodel.add(Activation('relu'))
Newmodel.add(Conv2D(32, (3, 3)))
Newmodel.add(Activation('relu'))
Newmodel.add(MaxPooling2D(pool_size=(2, 2)))
Newmodel.add(Dropout(0.25))

Newmodel.add(Conv2D(64, (3, 3), padding='same'))
Newmodel.add(Activation('relu'))
Newmodel.add(Conv2D(64, (3, 3)))
Newmodel.add(Activation('relu'))
Newmodel.add(Conv2D(64, (3, 3)))
Newmodel.add(Activation('relu'))
Newmodel.add(MaxPooling2D(pool_size=(2, 2)))
Newmodel.add(Dropout(0.25))

Newmodel.add(Flatten())
Newmodel.add(Dense(512))
Newmodel.add(Activation('relu'))
Newmodel.add(Dense(128))
Newmodel.add(Activation('relu'))
Newmodel.add(Dense(64))
Newmodel.add(Activation('relu'))
Newmodel.add(Dropout(0.5))
Newmodel.add(Dense(4))
Newmodel.add(Activation('softmax'))

opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)
Newmodel.compile(loss=keras.losses.categorical_crossentropy,
    ↪ optimizer=opt,metrics=['accuracy'])

#Cargamos los pesos de la CNN
Newmodel.load_weights('keras_faceRecognition_CIFAR10_233_94_.h5')

# Load image
img_path = "image2.jpg"
image = io.imread(img_path)

def detect_faces(image):

    # Create a face detector
    face_detector = dlib.get_frontal_face_detector()

    # Run detector and get bounding boxes of the faces on image.
    detected_faces = face_detector(image, 1)
    face_frames = [(x.left(), x.top(),x.right(), x.bottom()) for x in
        ↪ detected_faces]

    return face_frames

# Detect faces
```

```
79 detected_faces = detect_faces(image)
80
81 # Crop faces and plot
82 for n, face_rect in enumerate(detected_faces):
83     face = Image.fromarray(image).crop(face_rect)
84
85 #TEST
86 test_image = face
87 test_image = test_image.resize((152, 152))
88 test_image=im.img_to_array(test_image)
89 test_image=np.expand_dims(test_image, axis=0)
90
91 result=Newmodel.predict(test_image)
92
93 #RESULTADOS
94 print(result)
95 if result[0][0]==1.0:
96     print('Sujeto 1')
97 elif result[0][1]==1.0:
98     print('Sujeto 2')
99 elif result[0][2]==1.0:
100    print('Sujeto 3')
101 elif result[0][3]==1.0:
102    print('Sujeto 4')
```

Código fuente 2.2: Clasificacion mediante CNN entrenada

Lista de tablas

Lista de figuras

Lista de códigos fuente

Bibliografía

[1] *Apliación de las redes neuronales en medicina* Alberto Delgado. Universidad Nacional de Colombia 1999.

[2] *End to End Learning for Self-Driving Cars* Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba. Cornell University 2016

[3] *TensorFlow.* Biblioteca desarrollada por Google para Machine Learning. Link: https://www.tensorflow.org

[4] *Keras.* Biblioteca desarrollada para usar junto con TensorFlow, permitiendo un mayor nivel de abstracción. Link: https://keras.io

[5] *10k US Adult Faces Database* Base de datos con 10,168 imágenes de caras de 2,222 sujetos. http://www.wilmabainbridge.com/facememorability2.html

[6] *AT & T - The Database of Faces* Base de datos con imágenes de caras de 40 sujetos entre abril de 1992 y abril de 1994 . https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

[7] *Caltech Faces* Base de datos con 450 imágenes de caras de 27 sujetos. http://www.vision.caltech.edu/html-files/archive.html

[8] MIT-CBCL *Component-based Face Recognition with 3D Morphable Models, First IEEE Workshop on Face Processing in Video* B.Weyrauch, J.Huang, B.Heisele, and V.Blanz, Washington D.C., 2004. Credit is hereby given to the Massachusetts Institute of Technology and to the Center for Biological and Computational Learning for providing the database of facial images.Copyright 2003 -2005 Massachusetts Institute of Technology. All Rights Reserved.

[9] *ResNet50 pre-entrenada* Red neuronal convolucional ResNet50 previamente entrenada.

[10] *VGGnet16 pre-entrenada* Red neuronal convolucional VGGnet16 previamente entrenada. https://github.com/fchollet/deep-learning-models

[11] *Google Colaboratory.* Plataforma en la nube de Google para ejecutar código en Python 2 y 3. Link: https://colab.research.google.com/

[12] *DeepFace, Closing the Gap to Human-Level Performance in Face Verification.* Yaniv Taigman, Ming Yang, Marc Aurelio Ranzato and Lior Wolf Menlo Park, CA, USA, 2014.

[13] *Gradient-Based Learning applied to document recognition* Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner. Proc. of the IEEE, November 1998.

[14] *Very Deep Convolutional Networks for Large-Scale Image Recognition* Karen Simonyan, Andrew Zisserman. Cornell University, 4th of September 2014.

[15] *Deep Residual Learning for Image Recognition* Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Cornell University, 10th of December 2015.

[16] *Wide Residual Networks* Sergey Zagoruyko, Nikos Komodakis. Cornell University, 23th of May 2016.

[17] *Densely Connected Convolutional Networks* Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. Cornell University, 25th of August 2016.

[18] *ImageNet Classification with Deep Convolutional Neural Networks* Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. University of Toronto, 2012.