UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación de reconocimiento en imágenes utilizando *Deep Learning* con OpenCV

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Oliva Rodríguez, Alejandro

Tutor: Agustí Melchor, Manuel

Curso 2017-2018

Resumen

Las recientes técnicas de aprendizaje de modelos basadas en aprendizaje profundo (*Deep Learning*), en visión por computador, prometen altas tasas de reconocimiento. Pero el proceso de elaboración de los conjuntos de imágenes de entrenamiento y test, el número y complejidad de redes que los implementan y el número de parámetros a evaluar hace difícil su utilización de manera contrastada.

Este trabajo consiste en el desarrollo de una herramienta que asiste a la creación de esos modelos de aprendizaje profundo cuyos resultados puedan ser empleados para la puesta en práctica en problemas enfocados a la visión por computador.

El desarrollo de la herramienta ha de permitir al usuario la creación y manipulación de un conjunto de datos de forma gráfica, la creación y entrenamiento de una red neuronal convolucional y la posibilidad de probar la efectividad de los diferentes modelos entrenados. Para la implementación ha sido necesario la utilización de múltiples tecnologías y la implantación de diferentes comunicaciones entre estas.

Por último se plantea un problema completo de reconocimiento de objetos donde la herramienta implementada servirá para la resolución de una parte de este. Se ha buscado un problema con restricciones temporales importantes para estudiar el uso de los reconocedores y evaluar la necesidad de introducir otras técnicas y mejoras que permitan su uso en aplicaciones con fuertes restricciones temporales. La segunda parte del problema se proponen diferentes métodos utilizados en la visión por computador para la detección de objetos, donde, para cada método utilizado se detalla la implementación y los resultados obtenidos.

Palabras clave: Aprendizaje profundo, *Deep Learning*, Redes neuronales convolucionales, Keras,Visión por computador, Detección de objetos, OpenCV

Resum

Les recents tècniques d'aprenentatge de models basades en aprenentatge profund (Deep Learning), en visió per computador, prometen altes taxes de reconeixement. Però el procés d'elaboració dels conjunts d'imatges d'entrenament i test, el nombre i complexitat de xarxes que els implementen i el nombre de paràmetres a avaluar fa difícil la seva utilització de manera contrastada.

Aquest treball consisteix en el desenvolupament d'una eina que assisteix a la creació d'eixos models d'aprenentatge profund on els resultats puguin ser emprats per a la posada en pràctica en problemes enfocats a la visió per computador.

El desenvolupament de l'eina ha de permetre a l'usuari la creació i manipulació d'un conjunt de dades de forma gràfica, la creació i entrenament d'una xarxa neuronal convolucional i la possibilitat de provar l'efectivitat dels diferents models entrenats. Per a la implementació ha estat necessari la utilització de múltiples tecnologies i la implantació de diferents comunicacions entre aquestes.

Finalment es planteja un problema complet de reconeixement d'objectes on l'eina implementada servirà per a la resolució d'una part d'aquest. S'ha buscat un problema amb restriccions temporals importants per estudiar l'ús dels reconeixedors i avaluar la necessitat d'introduir altres tècniques i millores que permetin el seu ús en aplicacions amb fortes restriccions temporals. La segona part del problema es proposen diferents mètodes utilitzats en la visió per computador per a la detecció d'objectes, on, per a cada mètode utilitzat es detalla la implementació i els resultats obtinguts.

Paraules clau: Aprenentatge profund, *Deep Learning*, Xarxes neuronals convolucionals, Keras, Visió per computador, Detecció d'objectes, OpenCV

Abstract

The recent learning techniques of models based on Deep Learning, in computer vision, promise high recognition rates. But the process of preparing the sets of training and test images, the number and complexity of the networks that implement them and the number of parameters to be evaluated make it difficult to use them in a contrasted way.

This work consists of the development of a tool based on deep learning models whose results can be used for the implementation of problems focused on computer vision.

The development of the tool should allow the user to create and manipulate a set of data in a graphical way, the creation and training of a convolutional neuronal network and the possibility of testing the effectiveness of the different trained models. For the implementation has been necessary the use of multiple technologies and the implementation of different communications between them.

Finally, there is a complete problem of objects recognition where the implemented tool will serve for the resolution of a part of it. A problem with important temporal restrictions has been sought to study the use of recognizers and to evaluate the need to introduce other techniques and improvements that allow their use in applications with severe temporal restrictions. The second part of the problem proposes different methods used in computer vision for the detection of objects, where, for each method used, the implementation and the results obtained are detailed.

Key words: *Deep Learning*, Convolutional Neural Networks, Keras, Computer Vision, Object detection, OpenCV

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Los resultados de las recientes técnicas de aprendizaje automático (*Machine Learning* un campo de la inteligencia artificial) está alcanzando actualmente límites impensables hace apenas 10 años atrás. Las principales empresas y gobiernos del mundo están invirtiendo numerosas cantidades de dinero para la investigación en este campo[1][2].

En la visión por computador se iniciaron las primeras investigaciones alrededor de los años 50. Desde aquella época han surgido diferentes técnicas a partir de las cuales se han ido mejorando los resultados. Sin embargo con la llegada de las redes neuronales profundas se ha conseguido llevar este campo a un nivel superior. La irrupción de las redes neuronales profundas, más específicamente las redes neuronales convolucionales, está permitiendo el desarrollo de aplicaciones de visión por computador como por ejemplo el desarrollo de coches autónomos.

Esto puede dar a entender que las nuevas técnicas de *Deep Learning* actúan por arte de magia resolviendo problemas impensables años atrás. Pero nada más lejos de la realidad, para conseguir estos resultados se han de tener conjuntos de datos extensos con los que entrenar a los clasificadores. Si es cierto que en la actualidad existen más facilidades para conseguir estos conjuntos de datos o reutilizar conjuntos de datos ya creados. Para el procesamiento de estos grandes conjuntos de datos también ha sido relevante el avance de las unidades de procesamiento gráfico (GPU), gracias a ellas se han podido reducir considerablemente los tiempos de procesamiento de los modelos de red. Sin embargo, muchas veces el entrenamiento de un conjunto de datos pueda llevar días o incluso semanas.

El presente trabajo final de grado o TFG presenta una herramienta desarrollada para facilitar la creación de un conjunto de datos, el diseño y entrenamiento de una red neuronal convolucional. A partir de los resultados obtenidos se aborda un problema práctico y completo que abarca la utilización de técnicas de visión por computador.

1.1 Motivación

La continua evolución de los diferentes campos de la inteligencia artificial, las numerosas aplicaciones que aparecen cada día. Lo que antes parecía algo futurista, se está convirtiendo en el futuro más próximo. Las empresas cada vez invierten más en este campo para solucionar de forma eficiente sus problemas. Sin lugar a duda, esto no es el trabajo del futuro, es el trabajo del presente.

Durante los dos últimos años de carrera se ha cursado la rama de Computación donde se introduce brevemente al mundo de las redes neuronales y a la visión por computador. La elección de esta rama se debe al actual auge que está teniendo la inteligencia artificial

en la sociedad y a la curiosidad por entender cómo se consiguen y cómo trabajan las principales aplicaciones en este campo.

No tener experiencia en estos campos, se ha convertido en una meta personal para la elección de este trabajo y realizar un proceso de aprendizaje de forma autodidacta. No es una tarea sencilla y por esta razón se hace una selección de diferentes cursos que puedan ser de ayuda. La plataforma de educación virtual Coursera [3], ofrece cursos sobre redes neuronales profundas y redes neuronales convolucionales de la mano de Andrew Ng que a su vez es uno de los fundadores de la plataforma. Para la parte de visión por computador, se encuentra el *blog* de Adrian Rosebrock [4], el cual ofrece una amplia gama de ejemplos muy valorados por la comunidad dedicada a este campo.

1.2 Objetivos

El primer objetivo de este trabajo es el desarrollo de una herramienta que permita al usuario la creación de una red convolucional y el posterior entrenamiento de la red creada sobre un conjunto de datos. Este objetivo principal se descompone en los siguientes subobjetivos:

- Permitir la creación, modificación y carga de un conjunto de datos de imágenes.
- Crear un modelo de red neuronal convolucional.
  - A partir de un modelo de red existente.
  - Nuevo modelo creado por el usuario.
- Entrenar un modelo de red creado con la herramienta.
  - Monitorizar resultados del entrenamiento.
  - Guardar resultados y modelo entrenado.
- Cargar resultados y modelo de red entrenado.
  - Realizar pruebas *test* sobre el modelo entrenado.

Es necesario la consecución de todos los subobjetivos para finalmente alcanzar el objetivo global correspondiente a la creación de una herramienta basada en modelos de aprendizaje profundo.

Como último objetivo, se propone la creación de un conjunto de datos y una red neuronal convolucional para realizar pruebas con la herramienta creada. Los resultados se aplicarán a un ejemplo completo donde se tratan diferentes técnicas de la visión por computador.

1.3 Estructura de la memoria

El presente trabajo está compuesto por 5 capítulos. A continuación se menciona la información que se puede encontrar en cada uno de estos:

- **Capítulo 1. Introducción:** en el primer capítulo se hace una introducción al tema que se tratará durante todo el proyecto, los objetivos a alcanzar y las motivaciones que han dado paso al desarrollo del trabajo.

- **Capítulo 2. Fundamentos teóricos:** En este capítulo se introduce al lector de forma teórica sobre los campos de la visión por computador y el aprendizaje profundo.
- **Capítulo 3. Herramienta para la definición de un modelo basada en aprendizaje profundo:** Se exponen los requisitos de la herramienta propuesta para desarrollar. Las tecnologías empleadas para el desarrollo, la arquitectura general de la aplicación, la funcionalidad e implementación de cada módulo de la herramienta y los usuarios finales de la aplicación.
- **Capítulo 4. Problema planteado:** Se propone un problema a resolver a partir del cual se desarrollan diferentes soluciones. El problema sirve para realizar pruebas sobre la herramienta explicada en el tercer capítulo y aplicar técnicas utilizadas en la visión por computador.
- **Capítulo 5. Conclusiones:** en el último capítulo se aportan las conclusiones sobre el trabajo realizado. También se plantean los trabajos futuros a desarrollar en la herramienta.

Se añade una bibliografía con las referencias consultadas durante la realización del trabajo. Por último, una serie de apéndices concluyen el trabajo.

CAPÍTULO 2
Estado del arte y fundamentos teóricos

En este capítulo se hace una introducción al concepto de visión por computador así como a las técnicas que se han estado empleando hasta la fecha y que están siendo reemplazadas por las redes neuronales profundas. Así mismo, se hace más hincapié en las redes neuronales convolucionales donde se explica de forma más detallada el funcionamiento de estas.

2.1 Visión por computador

Como su nombre indica, el principal objetivo de la visión por computador es intentar imitar la funcionalidad del sistema de visión. Acciones como diferenciar un coche o diferenciar objetos visibles en un fondo son acciones que las personas realizan a diario. Esto es una tarea compleja la cual lleva años de investigación para que un ordenador sea capaz de detectarlo con una precisión razonable.

La visión por computador está obteniendo unos avances importantes en los últimos años. La puesta en escena de la redes neuronales profundas y más concretamente de las redes neuronales convolucionales han dado paso a este salto de nivel. El reconocimiento facial, de señales de tráfico o los coches autónomos son algunos de los ejemplos de las aplicaciones que se están consiguiendo.

Sin embargo, antes de llegar a este punto hay un largo recorrido que es importante mencionar para ver la evolución que ha sufrido la visión por computador. En el año 2001 apareció el algoritmo Viola-Jones capaz de detectar rostros de forma eficiente[5]. El algoritmo detecta rostros en tiempo real a partir de una cámara web, en aquel momento fue considerada una de las mejores aportaciones al mundo de la visión por computador. Pocos años después, en 2005, apareció *Histogram of oriented gradients* (HOG)[7]. Este nuevo método superó a los algoritmos existentes para la detección de personas[8].

Para la clasificación de imágenes se hace uso de los métodos de extracción de características mencionados y posteriormente de un algoritmo de clasificación para las características extraídas. No obstante, antes de aplicar estos pasos se realiza un preproceso a la imagen para normalizar los efectos de contraste y brillo y la eliminación de ruido (ver Figura 2.1) [8].

En los dos siguientes subapartados se repasan algunas de las principales técnicas que se han estado utilizando para la extracción de características y la clasificación de estas.

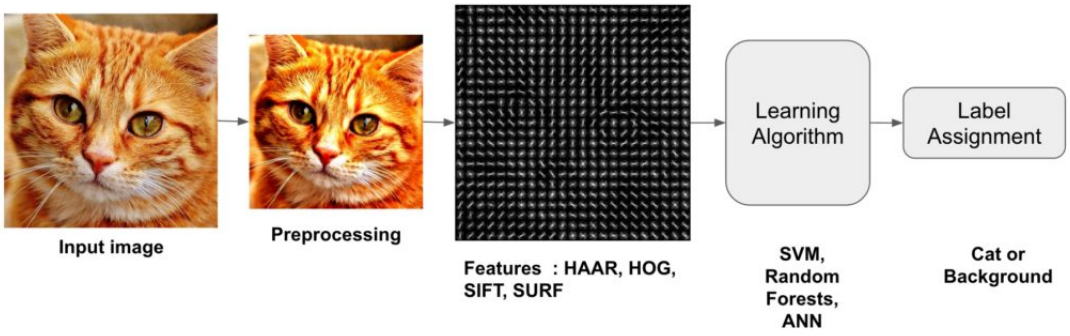

Figura 2.1: Pasos de un clasificador de imagen. Imagen extraída de [8].

2.1.1. Extracción de características

Una imagen de entrada contiene una gran cantidad de características. Sin embargo una gran multitud de estas pueden ser irrelevantes y negativas para el paso de la clasificación. Por ello, el primer paso a realizar es la simplificación de la imagen, extrayendo de estas la información importante y eliminando aquella que no proporciona información para la clasificación. A este proceso se le llama extracción de características[8]. Los siguientes métodos son algunos de los que se utilizaban o utilizan para realizar este proceso.

Filtros en cascada

Paul Viola y Michael Jones propusieron un método para la detección de objetos utilizando clasificadores en cascada basados en características Haar [5]. Partiendo de un gran conjunto de imágenes positivas y negativas, se aplican los denominados filtros de Haar (ver Figura 2.2). Cada una de las ventanas se aplica en la imagen extrayendo una única característica, cuyo valor se obtiene a partir de la diferencia en la suma de los valores de los píxeles entre zonas en negro y zonas en blanco. Cuando se extraen todas las características de una imagen muchas de ellas son irrelevantes. Para eliminar estas se aplica la técnica Adaboost que consiste en la utilización de clasificadores débiles para alcanzar a partir de estos un clasificador fuerte [9]. Una de las principales aplicaciones donde se aplica la técnica de filtros en cascada es para la detección de rostros.

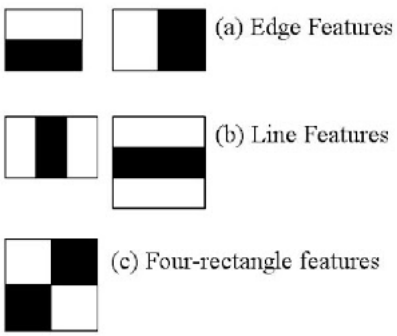

Figura 2.2: Filtros de Haar. Imagen extraída de [9].

LBP (*Local Binary Patterns*)

LBP es un descriptor de texturas introducidos por Ojala et. al. que calcula una representación local de la textura que se construye comparando cada píxel con los píxeles vecinos[6]. Si la intensidad de un píxel vecino es superior al píxel central, entonces se aplica el valor 0 para el píxel vecino, en caso contrario será 1 [10].

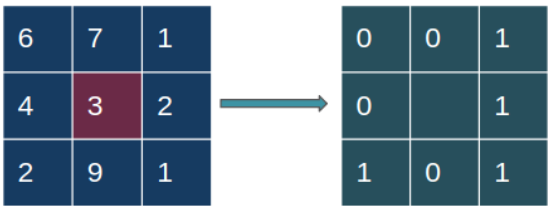

Figura 2.3: Descriptor LBP que opera en un vecindario de píxeles fijo de 3 x 3.

Cuando se obtiene el resultado en binario, se establece un orden y se calcula el valor del píxel central como se observa en la Figura 2.4. Es importante mantener el orden establecido para realizar la suma para los siguientes píxeles. Por último se calcula el histograma de características LBP a partir del cual se pueden detectar patrones en una textura y a raíz de ello realizar la clasificación de una imagen con algún algoritmo de clasificación [10].

Figura 2.4: Test binario almacenado en una lista de 8-bits.

HOG (*Histogram of Oriented Gradients*)

El método HOG fue propuesto en 2005 por Dalal y Triggs y demostró que el método HOG en combinación con máquinas de vectores soporte podía utilizarse para entrenar clasificadores altamente precisos [7]. HOG divide una imagen en ventanas realizando un configuración en rejilla. De cada ventana se obtiene un histograma donde las entradas son las orientaciones de los gradientes [11].

Como otros métodos, HOG es un extractor de características lo que se podría entender que transforma una imagen a una versión comprimida de esta. Para diferenciar conjuntos de características hace falta un algoritmo que sea capaz de separarlos, aquí es donde se emplean las máquinas de vectores soporte.

2.1.2. Clasificadores

Con los métodos descritos en el apartado anterior, se explica el proceso de cómo convertir una imagen a un vector de características. Los algoritmos de aprendizaje automático son los encargados de tomar como entrada este vector de características y producir

como salida una etiqueta de clase. En la actualidad hay una gran variedad de algoritmos de . En la Figura 2.5 se puede observar una clasificación de algunos de los algoritmos más utilizados en el aprendizaje automático.

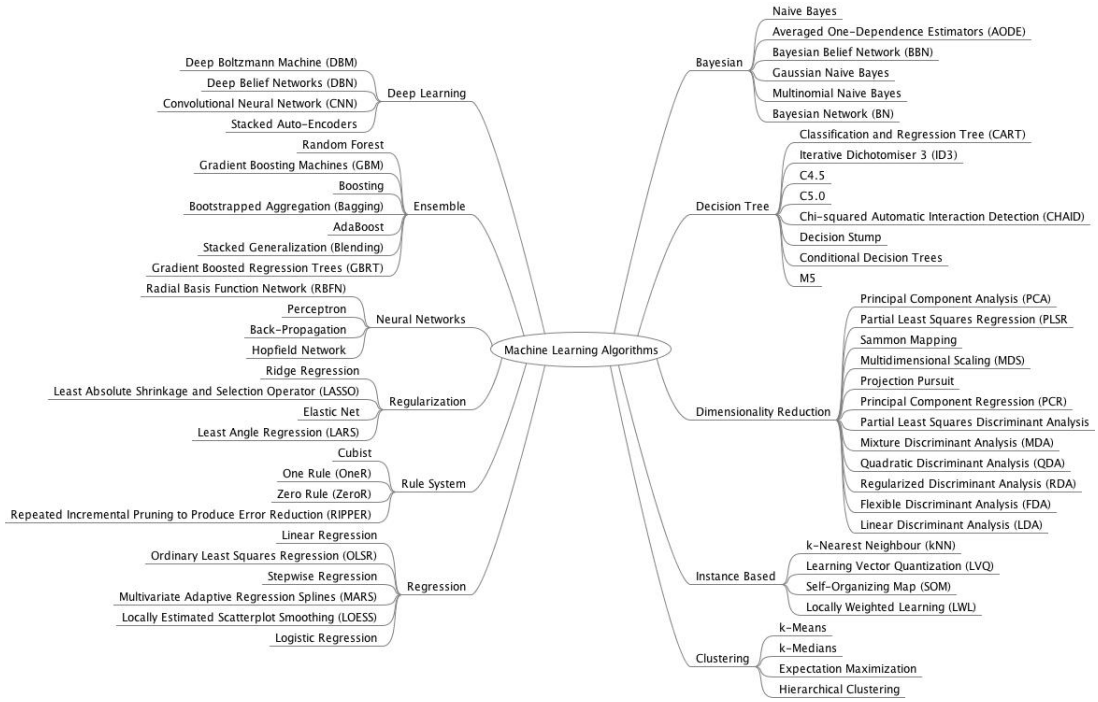

Figura 2.5: Tipos de algoritmos de aprendizaje automático. Imagen extraída de [12]

Los algoritmos de aprendizaje automático se clasifican en diferentes tipos [12], entre otros:

- Aprendizaje supervisado o *Supervised Learning*.
- Aprendizaje no supervisado o *Unsupervised Learning*.
- Aprendizaje semi supervisado o *Semi-supervised. Learning*

Aprendizaje supervisado

En el aprendizaje supervisado antes de realizar una clasificación para una imagen se realiza una preparación. Esto consiste en preparar un conjunto de datos perfectamente etiquetado[13]. En términos de visión por computador todas las imágenes que que conforman el conjunto de datos han de contener el objeto con la finalidad de que el clasificador aprenda las características del objeto a clasificar. A partir del resultado obtenido, el clasificador es capaz de encontrar los objetos clasificados en otras imágenes diferentes.

Existen diferentes tipos de algoritmos que emplean este tipo de aprendizaje. Uno de los más utilizados para la visión por computador después de aplicar técnicas para la extracción de características como HOG son las máquinas de vectores soporte o SVM. Este tipo de aprendizaje también se emplea para los apartados que se describen más adelante sobre las redes neuronales profundas.

Aprendizaje no supervisado

A diferencia que en el aprendizaje supervisado, existe un conjunto de datos pero el cual no está etiquetado. El clasificador es el encargado de realizar la clasificación a partir de las características que reciba en la entrada[13].

Una de las principales funciones, es realizar agrupaciones (*clustering*) a partir de las características. Estos agrupamientos se realizan a partir de características que sean muy similares entre sí. Algunos de los algoritmos de clasificación utilizados en este tipo de aprendizaje son *K-means* o *PCA (Principal Component Analysis)* entre otros.

Aprendizaje semi supervisado

Este tipo de aprendizaje es una mezcla de los dos anteriores. Se tiene un conjunto de datos donde una parte pequeña de este se encuentra etiquetada y el resto sin etiquetar. Se suelen utilizar técnicas como el agrupamiento vistas en el aprendizaje no supervisado, con el fin de observar con qué grupo etiquetado se relacionan las muestras no etiquetadas[13].

En el Apéndice C se explica como realizar un entrenamiento con clasificadores Haar-Cascade, donde se ilustra en cierta manera, cómo se hace uso de técnicas similares a las mencionadas en este apartado.

2.2 *Deep Learning*

El aprendizaje profundo o *Deep Learning* es un subcampo del aprendizaje automático o *Machine Learning*, que a su vez, es un subcampo de la inteligencia artificial (IA)[15].

El principal objetivo de la IA es ofrecer algoritmos y técnicas para resolver problemas que las personas pueden resolver de forma intuitiva o automatizada. Un ejemplo es como la IA puede interpretar el contenido de una imagen y extraer conclusiones sobre esta.

El aprendizaje profundo se divide en diferentes ramas como redes neuronales, redes neuronales convolucionales o redes neuronales recurrentes. Estas arquitecturas son utilizadas en campos como la visión por computador, reconocimiento del habla o reconocimiento de dígitos manuscritos entre otros.

2.2.1. Redes neuronales

Las redes neuronales son un modelo computacional basado en las neuronas del sistema biológico, con la finalidad de reproducir, de forma aproximada, el comportamiento de estas [16]. Funciona con un elevado número de unidades interconectadas nombradas neuronas.

Las neuronas se organizan en capas. Normalmente una red neuronal está compuesta por tres partes: la capa de entrada, con neuronas que representan los campos de entrada; una o más capas ocultas; y una capa de salida con una o más unidades que representan el resultado computado por la red [14]. En la primera capa se introducen los datos de entrada, y los valores se van propagando desde cada neurona hasta las neuronas de la capa siguiente. Finalmente la capa de salida devuelve un resultado. Las conexiones entre neuronas están representadas por un peso, que puede ser positivo o negativo dependiendo si una neurona activa o inhibe a otra (ver Figura 2.6) [16].

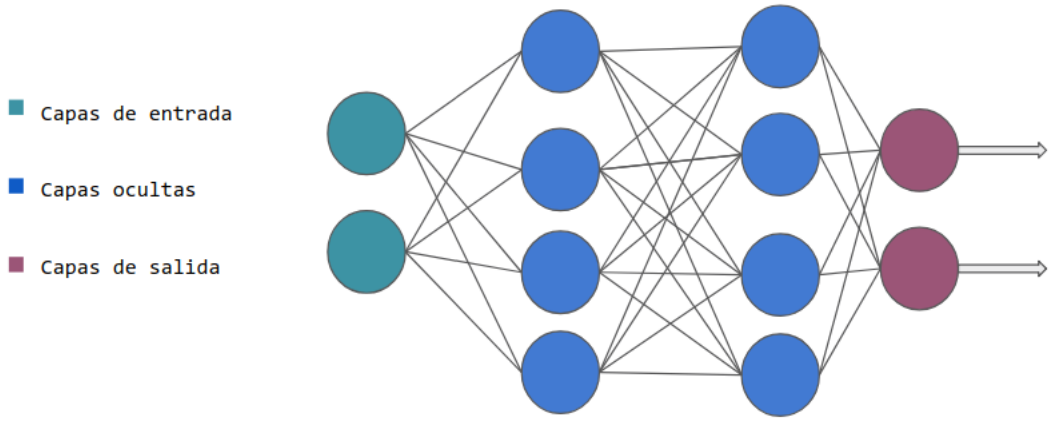

Figura 2.6: Red neuronal multicapa.

La red aprende a partir de un proceso de retroalimentación llamado *Backpropagation*. Consiste en comparar el resultado obtenido en la capa de salida con el resultado que se pretendía obtener[17]. Con la diferencia entre ambos resultados se modifican los pesos de las capas ocultas hasta la capa de entrada. Después de varias iteraciones la red empieza a aprender y reduce la diferencia entre el resultado esperado y obtenido [17].

2.2.2. Estado del arte

Hasta el año 2010 las redes neuronales habían visto reducir su uso por la comunidad científica. Pocas investigadores seguían trabajando en redes neuronales entre ellos: los grupos de Geoffrey Hinton en la Universidad de Toronto, Yoshua Bengio en la Universidad de Montreal, Yann LeCun en la Universidad de Nueva York y IDSIA en Suiza [15].

En 2011 Dan Ciresan ganó una competición de clasificación de imágenes con algoritmos de redes neuronales profundas. Pero el momento clave llegaría un año después en 2012, con la aparición del grupo de Geoffrey Hinton en la competición ImageNet [18].

ImagNet fue una competición que consistía en resolver un problema de clasificación de imágenes en 1000 categorías después de haber realizado un entrenamiento sobre 1'4 millones de imágenes. Se consiguió una tasa de acierto del 83.6 %. Años más tarde en el 2015 se consiguió un acierto de 96.4 % considerando así que el problema estaba resuelto [18].

Especificando, las redes neuronales utilizadas para estas competiciones son llamadas redes neuronales convolucionales. Este tipo de redes son mayormente utilizadas para la visión por computador aunque también son empleadas para otros campos.

Gracias a los avances tecnológicos y tener una capacidad de cálculo mayor a la de hace escasos años, las redes neuronales son ahora un tema de actualidad donde la comunidad científica ha vuelto con fuerza para seguir mejorando este campo cada día.

2.3 Redes convolucionales

La clasificación de imágenes a partir de una red neuronal como las vistas en la sección anterior puede ser un problema debido a la carga computacional que esto supone. Como se comenta, en las redes neuronales todas sus neuronas están totalmente conectadas entre

capas. Si por ejemplo se dispone de una imagen con unas dimensiones de 640 x 640 píxeles, esta hace un total de 409.600 datos de entrada que estarán conectados a cada una de las neuronas de la primera capa. Esto produce un gran número de cálculos que ralentizará el proceso de entrenamiento [14].

Para mitigar este problema se presentan las redes neuronales convolucionales. Las redes neuronales convolucionales están formadas por diferentes capas (ver Figura 2.7) [14], donde las primeras capas se encargan de extraer características como los bordes o esquinas de una imagen. Una vez detectados estos bordes son utilizados para detectar formas en las capas posteriores. Cuando se detectan las formas se procede a detectar las características de alto nivel como puede ser una rueda en una imagen donde aparece un coche. Las últimas capas están totalmente conectadas y son las encargadas de dar una predicción a partir de estas últimas características extraídas.

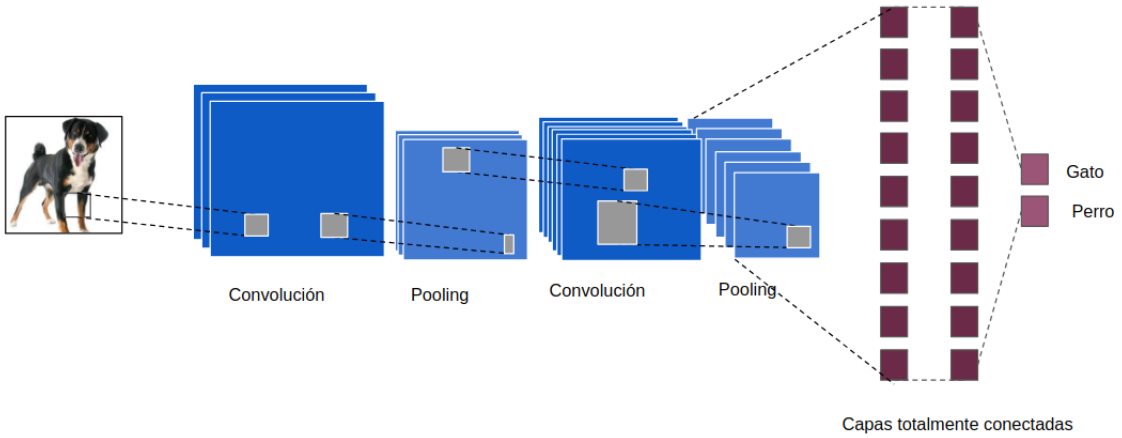

Figura 2.7: Red neuronal convolucional.

2.3.1. Convolución

Reducir el número de conexiones entre la capa oculta y la capa de entrada es su principal funcionalidad. A partir de unos filtros o también llamados *kernel* se extraen las principales características de una imagen [14].

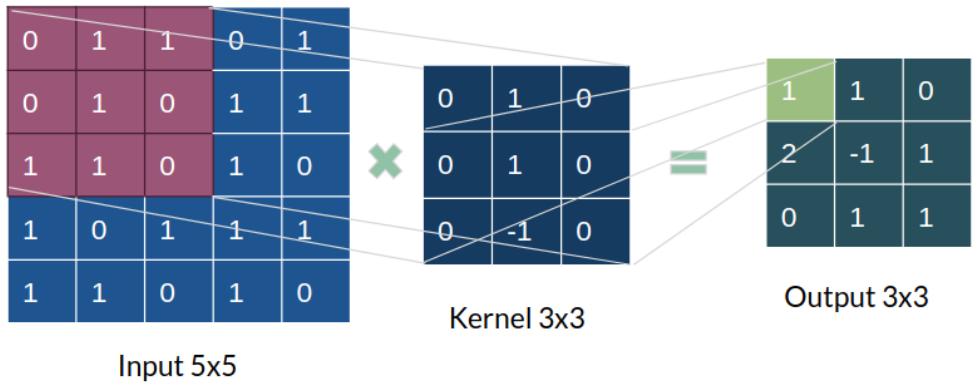

Figura 2.8: Aplicación de una operación de convolución.

En la Figura 2.8 se observa cómo se realiza un producto escalar entre el *kernel* y la matriz de la imagen de entrada. En esta imagen se ha establecido un único *kernel* pero a la hora de calcular las dimensiones de la salida se tiene en cuenta [19]:

- **E** = tamaño de la imagen de entrada.
- **K** = tamaño del *kernel* utilizado.
- **N** = número de *kernels* utilizados.
- **S** = *stride*, que indica el desplazamiento del *kernel* sobre la matriz de entrada.
- **M** = margen añadido a la matriz de entrada.
- **O** = dimensiones de salida

Para calcular la salida se realiza el siguiente cálculo [19]:

$$O = \frac{E - K + 2M}{S} \tag{2.1}$$

El número de canales de la imagen es igual a el número de *kernels* utilizados.

2.3.2. *Pooling*

Extraídas la características desde una capa de convolución la siguiente capa es la de *Pooling*. La funcionalidad es reducir las dimensiones de la salida de la capa convolucional [14]. Hay diferentes maneras de realizar esta reducción:

- ***Max Pooling***: se desliza un filtro de tamaño *m x m* a través de la matriz de entrada y se selecciona el valor más grande de dicha matriz (ver Figura 2.9) [14].
- ***Average Pooling***: igual que en la anterior se desliza un filtro, pero el resultado es la media de todos los elementos de la matriz de entrada [14].

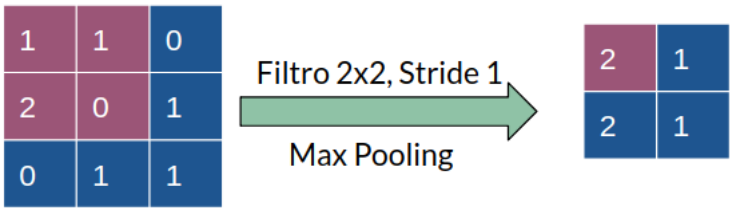

Figura 2.9: Operación *MaxPooling* con *stride* (2,2).

Para esta capa también se aplica un *stride* cómo en la capa de convolución. El número de canales no se reduce, únicamente el número de dimensiones.

2.3.3. ReLU

Rectified linear unit (ReLU) es una función de activación que se encarga de truncar los valores negativos a 0 [14]. Normalmente es aplicada en la salida de una capa convolucional con la finalidad de introducir la no linealidad en un sistema que ha calculado operaciones lineales durante la capa convolucional.

$$f(x) = \boldsymbol{max}(x, 0) \tag{2.2}$$

2.3.4. Capas totalmente conectadas

Las últimas capas de la red convolucional actúan como clasificador. La función de estas capas son las mismas que se explican en la sección de Redes Neuronales. En estas últimas capas es habitual aplicar la técnica de *Dropout* (ver Figura 2.10) que desactiva un número de neuronas para evitar el sobreajuste o *overfitting* [14].

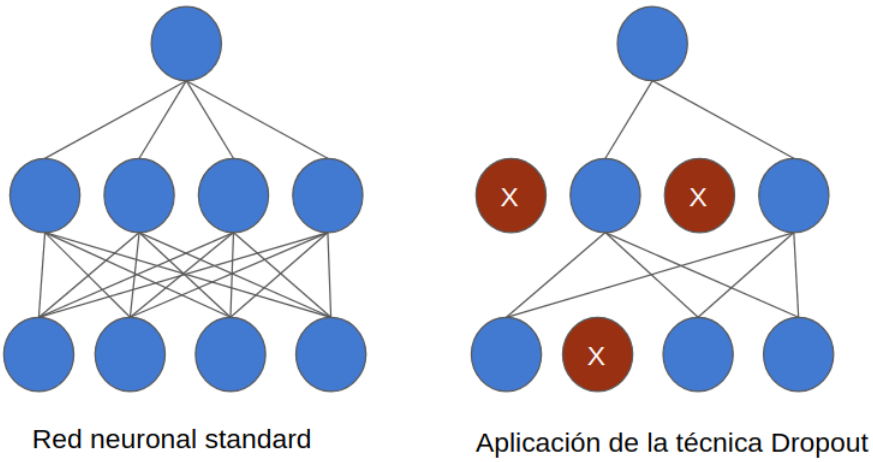

Figura 2.10: Aplicación de la técnica *Dropout*.

El número de neuronas en la última capa ha de ser igual al número de clases para los cuales ha sido entrenada la red.

2.4 Entrenamiento de un modelo

A la hora de realizar un entrenamiento para un modelo de red neuronal profunda existen diferentes formas para la puesta en marcha de el proceso de aprendizaje para la clasificación de objetos. Las mas habituales son el entrenamiento desde cero o la transferencia de aprendizaje [20].

Entrenamiento desde cero

Se necesita un amplio conjunto de datos correctamente etiquetado y un modelo de red que sea capaz de aprender características. Se emplean cuando se implementan modelos de red con un gran numero de salidas a clasificar. No suele ser común utilizar este tipo debido a que un gran conjunto de datos puede demorarse dias o semanas para realizar un entrenamiento [20].

Transferencia de aprendizaje

Este enfoque es uno de los mas empleados a la hora de desarrollar una nueva aplicación, ajustando detalladamente un modelo ya entrenado. Se selecciona un modelo de red entrenado y se ajusta al nuevo conjunto de datos, modificando el numero de salidas. Una de las principales ventajas es, que ya no se necesitan grandes conjuntos de datos, reduciendo así de forma considerable, el tiempo para el proceso de aprendizaje a minutos o horas [20]. En la Tabla 2.1 se observan algunas de las arquitecturas preentrenadas disponibles para utilizar con el enfoque de la transferencia de aprendizaje.

Modelo	Tamaño	Top-1 Accuracy	Top-5 Accuracy	Parámetros	Profundidad
Xception	88MB	0.790	0.945	22.910.480	126
VGG16	528MB	0.715	0.901	138.357.544	23
VGG19	549MB	0.727	0.910	143.667.240	26
ResNet50	99MB	0.759	0.929	25.636.712	168
InceptionV3	92MB	0.788	0.944	23.851.784	159
InceptionResNet	215MB	0.804	0.953	55.873.736	572
MobileNet	17MB	0.665	0.871	4.253.864	88
DenseNet121	33MB	0.745	0.918	8.062.504	121

Tabla 2.1: Documentación de modelos preentrenados en el conjunto ImageNet. Tabla extraída de [21].

Si el conjunto de datos con el que se entrenó el modelo de red es muy dispar con el conjunto que se va a realizar el entrenamiento, puede ser que no ofrezca resultados óptimos. En la Tabla 2.2 se plantean cuatro casos posibles en el uso de la transferencia de aprendizaje [22].

Casos	Conjunto de datos	Similitud con el conjunto de datos del modelo preentrenado
Caso 1	Pequeño	Alta
Caso 2	Pequeño	Baja
Caso 3	Grande	Alta
Caso 4	Grande	Baja

Tabla 2.2: Diferentes casos para el uso de la transferencia de aprendizaje.

En el Capítulo 4 se vuelve a mencionar este enfoque para realizar un entrenamiento, donde se explicará que Caso de la Tabla 2.2 se relaciona con el conjunto de datos utilizado para el entrenamiento.

2.5 Análisis de herramientas

Se realiza una búsqueda de herramientas que tengan un comportamiento similar a la que se desarrolla, con la finalidad de encontrar los puntos fuertes y débiles que puedan servir para mejorar el desarrollo. A continuación, se mencionan sin entrar en detalle las principales características para dos herramientas que tienen una finalidad similar a la herramienta que se presenta.

La primera herramienta es PipelineAi [23], una plataforma web que permite el entrenamiento de modelos de aprendizaje automático y aprendizaje profundo. Sus principales características son:

- Los resultados del entrenamiento de un modelo se sirven en tiempo real al usuario.
- Compatible con bibliotecas como Spark, Scikit-Learn, R, TensorFlow, Keras y Pythorch.
- Diferentes paneles para la gestión de los entrenamientos realizados.
- Ofrece diferentes plataformas *cloud* para acelerar el proceso de aprendizaje.
- Todos los servicios ofrecidos son de pago.

PipelineAi destaca por la cantidad de opciones que tiene a la hora de realizar un entrenamiento y por la compatibilidad con la mayoría de bibliotecas actuales dedicadas a los modelos basados en aprendizaje automático y profundo. Sin embargo, la utilización de servicios *cloud* hace que todas sus opciones sean de pago.

La segunda herramienta encontrada es Aetros [24], otra plataforma web que permite la monitorización en tiempo real de los resultados de un entrenamiento. Sus características principales son:

- Los resultados de un entrenamiento son servidos en tiempo real en diferentes gráficas que permiten al usuario una perfecta monitorización.
- Trabaja con las bibliotecas Keras y TensorFlow.
- Los entrenamientos son realizados en la máquina local del usuario o en un servicio *cloud*.
- Ofrece soporte para la creación de redes neuronales convolucionales.
- Se pueden comprobar los resultados de entrenamientos realizados con anterioridad.
- Tiene un servicio de prueba gratuito y el resto de pago.

Aetros presenta buenas características para el soporte de redes neuronales convolucionales. Su interfaz gráfica para la monitorización de los resultados en tiempo real están muy trabajada. Como punto "débil" presenta algunas restricciones a la hora de crear modelos de red complejos y para la creación de conjuntos de datos.

Estas dos herramientas brevemente analizadas, ofrecen buenas prestaciones para la puesta en marcha de el proceso de aprendizaje y su monitorización. Sin embargo, en la primera herramienta no se ha encontrado información relativa a la facilitación de la creación de conjuntos de datos. La segunda cumple con muchos de los objetivos propuestos para el desarrollo de la herramienta del presente trabajo, debido a que trabaja únicamente con modelos de redes neuronales convolucionales.

La herramienta que se implementa es una aplicación multiplataforma a diferencias de las herramientas analizadas que trabajan sobre plataformas web. Asiste a la creación de un conjunto de datos y a la modificación de este, mejorando en este punto a las anteriores herramientas. Por último, en la herramienta desarrollada, tanto el entrenamiento de un modelo como la monitorización se realizan en la máquina local del usuario aprovechando los recursos de esta y no en un servicio *cloud* externo.

CAPÍTULO 3
Herramienta para la definición de un modelo basado en aprendizaje profundo

Este capítulo aborda el diseño y desarrollo de una aplicación multiplataforma con la finalidad de facilitar al usuario una herramienta que le permita realizar entrenamientos de redes neuronales convolucionales. El usuario puede crear o cargar conjunto de datos de entrenamiento sobre el que realizar las tareas de entrenamiento. Los resultados son almacenados para posteriormente poder ser analizados, probados y extraer las conclusiones pertinentes.
En los siguientes apartados se detallan los requisitos de la herramienta tanto funcionales como no funcionales, las tecnologías empleadas para el desarrollo, descripción jerárquica y cada uno de los módulos de la herramienta.

3.1 Requisitos

En este apartado se detallan los requisitos funcionales y no funcionales. Los requisitos funcionales van a concordancia con los objetivos descritos en el primer capítulo del documento. A grandes rasgos, estos son los módulos que darán funcionalidad a la herramienta a desarrollar:

- Creación y carga de un conjunto de datos.
- Creación de una red convolucional.
  - Crear una red convolucional preentrenada.
  - Crear una red convolucional personalizada.
- Entrenamiento y monitorización de una red neuronal convolucional.
- Pruebas sobre una red entrenada.

3.1.1. Requisitos Funcionales

Para enunciar los requisitos se establece un identificador único para cada uno de los requisitos. Se añade un título y una descripción breve sobre el requisito establecido. Por último se indica a qué módulo de la herramienta está dirigido el requisito.

Identificador	RF-A-001
Título	Seleccionar directorio de entrenamiento y validación.
Descripción	El usuario tendrá la opción de seleccionar el directorio donde se añadirán las imágenes de entrenamiento y validación.
Módulo asociado	-Crear un conjunto de datos -Cargar un conjunto de datos

Identificador	RF-A-002
Título	Establecer el número de clases de un conjunto de datos.
Descripción	El usuario debe tener la opción de especificar el número de clases de un conjunto de datos. Por cada clases se ha generar una carpeta en los directorios de entrenamiento y validación
Módulo asociado	-Crear un conjunto de datos

Identificador	RF-A-003
Título	Acceder a cada clase del conjunto de datos.
Descripción	Se ha de ofrecer la posibilidad de acceder a cada una de las clases que se han creado con la finalidad de trabajar con cada una de forma individual.
Módulo asociado	-Crear un conjunto de datos -Cargar un conjunto de datos

Identificador	RF-A-004
Título	Añadir nombre a una clase.
Descripción	Se ha de poder especificar un nombre para cada una de las clases. Si el nombre es modificado la carpeta que se creó en los directorios de entrenamiento y validación ha de ser renombrada con el nombre de la clase.
Módulo asociado	-Crear un conjunto de datos -Cargar un conjunto de datos

Identificador	RF-A-005
Título	Añadir imágenes de entrenamiento y validación.
Descripción	El usuario ha de poder seleccionar una o más imágenes y añadirla al conjunto de datos. Se ha de separar esta opción para imágenes de entrenamiento y validación.
Módulo asociado	-Crear un conjunto de datos -Cargar un conjunto de datos

Identificador	RF-A-006
Título	Eliminar imágenes de entrenamiento y validación.
Descripción	El usuario ha de poder seleccionar una o más imágenes y eliminarlas del conjunto de datos. Se ha de separar esta opción para imágenes de entrenamiento y validación.
Módulo asociado	-Crear un conjunto de datos -Cargar un conjunto de datos

Identificador	RF-A-007
Título	Guardar una clase.
Descripción	El usuario ha de poder guardar una clase. Con esta opción se debe almacenar el nombre de la clase, número de imágenes de entrenamiento y validación y directorio donde se ubica la clase.
Módulo asociado	-Crear un conjunto de datos -Cargar un conjunto de datos

Identificador	RF-A-008
Título	Eliminar una clase.
Descripción	El usuario ha de poder eliminar una clase. Cuando se elimine una clase se han de eliminar todos los datos, carpetas e imágenes de la clase seleccionada.
Módulo asociado	-Crear un conjunto de datos -Cargar un conjunto de datos

Identificador	RF-A-009
Título	Visualizar imágenes de una clase.
Descripción	Se necesita implementar un tipo de galería donde se visualicen las imágenes de entrenamiento y validación de una clase. Si una imagen es seleccionada ha de estar disponible la opción de eliminar y editar imagen.
Módulo asociado	-Crear un conjunto de datos -Cargar un conjunto de datos

Identificador	RF-A-010
Título	Mostrar información de una clase.
Descripción	Es importante que la información como los directorios de una clase o como el número de imágenes de esta sean visibles para el usuario.
Módulo asociado	-Crear un conjunto de datos -Cargar un conjunto de datos

Identificador	RF-A-011
Título	Guardar conjunto de datos.
Descripción	Cuando el usuario haya definido un conjunto de datos dispondrá de la opción para guardar este conjunto. Ha de poder seleccionar el directorio donde ha de guardar la información del conjunto.
Módulo asociado	-Crear un conjunto de datos -Cargar un conjunto de datos

Identificador	RF-A-012
Título	Eliminar conjunto de datos.
Descripción	Si el usuario decide no guardar el conjunto de datos dispondrá de la opción de eliminar toda la información añadida durante el proceso de creación.
Módulo asociado	-Crear un conjunto de datos

Identificador	RF-A-013
Título	Seleccionar conjunto de datos.
Descripción	Para cargar un conjunto de datos el usuario debe seleccionar el fichero que guardo cuando creo un conjunto de datos.
Módulo asociado	-Cargar un conjunto de datos

Identificador	RF-B-001
Título	Seleccionar tipo de la red.
Descripción	El usuario tendrá dos opciones para crear una red. La primera opción debe ser elegir una red preentrenada o una segunda opción donde creará una red propia.
Módulo asociado	-Crear una red convolucional

Identificador	RF-B-002
Título	Seleccionar modelo de red preentrenada.
Descripción	Se han de ofrecer diferentes modelos preentrenados al usuario para que pueda trabajar con ellos.
Módulo asociado	-Crear una red convolucional -Crear una red convolucional preentrenada

Identificador	RF-B-003
Título	Seleccionar fichero con capas de salida.
Descripción	Cuando se selecciona una red preentrenada se ha de añadir una salida acorde al conjunto de datos del usuario. De esta manera se ha de añadir una opción para que el usuario introduzca un fichero con sus capas de salida.
Módulo asociado	-Crear una red convolucional -Crear una red convolucional preentrenada

Identificador	RF-B-005
Título	Seleccionar fichero con modelo completo.
Descripción	El usuario debe proporcionar un fichero que contenga un modelo de red convolucional.
Módulo asociado	-Crear una red convolucional -Crear una red convolucional personalizada

Identificador	RF-B-006
Título	Seleccionar un optimizador para la red.
Descripción	Para poner en marcha el procesos de aprendizaje de un un modelo de red neuronal es necesario elegir un optimizador y ajustar los parámetros de este. El usuario debe tener la opción de seleccionar un optimizador y poder completar los parámetros de este.
Módulo asociado	-Crear una red convolucional -Crear una red convolucional preentrenada -Crear una red convolucional personalizada

Identificador	RF-B-007
Título	Seleccionar una función de error.
Descripción	La función de error es la encargada de calcular el error entre el valor devuelto por la red y el valor esperado. Se deben incluir las funciones de error disponibles para que el usuario pueda seleccionar una de ellas.
Módulo asociado	-Crear una red convolucional -Crear una red convolucional preentrenada -Crear una red convolucional personalizada

Identificador	RF-B-010
Título	Guardar modelo de red.
Descripción	Una vez se complete la creación de una red se ha de permitir al usuario guardar los datos en un fichero para posteriormente a partir de este poder realizar un entrenamiento.
Módulo asociado	-Crear una red convolucional -Crear una red convolucional preentrenada -Crear una red convolucional personalizada

Identificador	RF-C-001
Título	Seleccionar fichero correspondiente a una red.
Descripción	El usuario ha de proporcionar a la herramienta el fichero donde se encuentran los datos de una red.
Módulo asociado	-Entrenamiento y monitorización

Identificador	RF-C-002
Título	Seleccionar fichero correspondiente a un conjunto de datos.
Descripción	El usuario ha de seleccionar el fichero donde se encuentren la información de un conjunto de datos.
Módulo asociado	-Entrenamiento y monitorización

Identificador	RF-C-003
Título	Iniciar el entrenamiento.
Descripción	Una vez se ha elegido el modelo de red y el conjunto de datos el usuario tendrá la opción de iniciar el entrenamiento.
Módulo asociado	-Entrenamiento y monitorización

Identificador	RF-C-004
Título	Guardar resultados.
Descripción	Cuando finaliza el entrenamiento, se debe activar una opción que permite guardar los resultado para posteriormente cargarlos en el módulo de pruebas.
Módulo asociado	-Entrenamiento y monitorización

Identificador	RF-C-005
Título	Monitorizar de forma gráfica, en tiempo real, los resultados del entrenamiento.
Descripción	Al final de cada iteración, la red devuelve un resultado sobre la tasa de acierto y de error. Los resultados se deben ir añadiendo a una gráfica para que el usuario pueda visualizarlos.
Módulo asociado	-Entrenamiento y monitorización

Identificador	RF-C-006
Título	Detener el proceso de entrenamiento.
Descripción	El usuario ha de poder detener en cualquier momento el proceso de entrenamiento de un modelo.
Módulo asociado	-Entrenamiento y monitorización

Identificador	RF-D-001
Título	Seleccionar fichero de resultados.
Descripción	Para mostrar los resultados de un entrenamiento se ha de cargar el fichero que se guardó al final de un entrenamiento. El usuario tendrá la opción de seleccionar el fichero mencionado.
Módulo asociado	-Entrenamiento y monitorización

Identificador	RF-D-002
Título	Seleccionar imagen.
Descripción	Se ha de permitir al usuario seleccionar una imagen para comprobar en qué clase se clasifica esta y así comprobar la efectividad de la red entrenada.
Módulo asociado	-Entrenamiento y monitorización

3.1.2. Requisitos no funcionales

Para enunciar los requisitos se establece un identificador único para cada uno de los requisitos. Se añade un título y una descripción breve sobre el requisito establecido.

Identificador	RNF-A-001
Título	Multiplataforma.
Descripción	La herramienta ha de ser compatible con sistemas Unix, MacOS y Windows.

Identificador	RNF-A-002
Título	Adaptar a resolución de pantalla.
Descripción	La herramienta se ha de adaptar a cualquier resolución de pantalla para ofrecer una buena experiencia de usuario.

Identificador	RNF-A-003
Título	Notificar errores.
Descripción	La herramienta ha de proporcionar mensajes de error informativos y orientados al usuario.

Identificador	RNF-A-004
Título	Tasa de aprendizaje.
Descripción	El usuario ha de aprender a utilizar la herramienta en un tiempo menor a una hora.

3.2 Metodología

En este breve apartado se menciona el paradigma para la implementación del proyecto. A partir de los requisitos,objetivos del proyecto y el uso de tecnologías web que son mencionadas en el siguiente apartado, se decide utilizar un paradigma orientado a eventos.

La programación dirigida por eventos es un paradigma de programación donde la ejecución del programa va determinada por los eventos provocados por el usuario o el propio sistema. Es trabajo del desarrollador definir los eventos que manejan el programa y las acciones a realizar cuando se produce un evento. Cuando empieza la ejecución de un programa dirigido por eventos se inicializa su código y queda a la espera de que se produzca algún evento. Un evento puede ser la acción de pulsar un determinado botón en la interfaz de usuario o la notificación de que se ha realizado una conexión a un servidor correctamente.

En el desarrollo de la herramienta se han implementado diferentes eventos que son inicializados cuando empieza la ejecución del programa. Sin embargo, también se implementan eventos de forma dinámica lo cual ha sido un gran punto a favor para la elección de este paradigma.

Sin nombrar todas las tecnologías utilizadas, se anticipa que se establece una comunicación entre diferentes tecnologías a consecuencia de la interacción del usuario con la interfaz. Estás tecnologías se dividen entre las que conforman la interfaz de usuario (tecnologías web) y las empleadas para trabajar con modelos de aprendizaje profundo

(tecnologías implementadas en Python). Aunque el usuario no interactúa directamente con las tecnologías externas de la herramienta (tecnologías Python), en estás también se producen eventos que se han de capturar en la interfaz. En la Figura 3.1 se presenta un esquema básico sobre el comportamiento de los eventos en la herramienta.

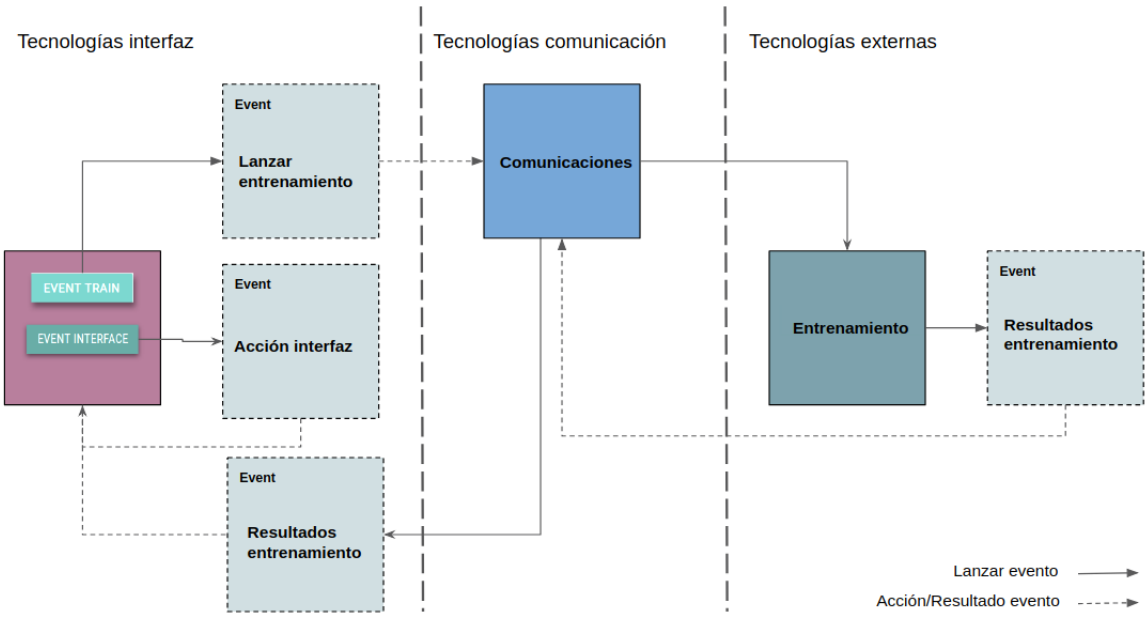

Figura 3.1: Esquema básico de la interacción eventos en la herramienta.

En el siguiente apartado se da a entender el motivo de la elección de el paradigma orientado a eventos a raíz de las tecnologías utilizadas y la necesidad de realizar las comunicaciones entre las diferentes tecnologías.

3.3 Tecnologías empleadas

A continuación se exponen las principales características de las tecnologías utilizadas y el motivo de la elección de cada una de ellas.

Electron

Electron es una biblioteca de código abierto desarrollada por GitHub para crear aplicaciones de escritorio multiplataforma con HTML, CSS y JavaScript. Electron logra esto combinando Chromium y Node.js en tiempo de ejecución y las aplicaciones se pueden ejecutar para Linux, Windows y Mac [25]. Se fundó en el año de 2013 al mismo tiempo que el editor de texto de Github, Atom. Un año después ambas aplicaciones fueron liberadas como código abierto [25].

Dispone de dos tipos de procesos siendo estos el proceso principal y el proceso renderizador. Solo pueda haber un proceso principal y este se encarga de ejecutar la aplicación y crear páginas web para la interfaz gráfica de usuario. Electron trabaja con Chromium[1] para mostrar páginas *webs* utilizando la arquitectura multiproceso, ejecutando así un proceso renderizador por cada página.

Una de las grandes ventajas que ofrece Electron es la introducción de Node.js en la arquitectura. Node.js puede ser utilizado tanto en el proceso principal como en el rende-

[1]Se trata de un proyecto de código abierto de navegador web.

rizador. Los usuarios pueden utilizar Node.js para acceder a recursos nativos del sistema operativo ampliando así las funcionalidades de las aplicaciones [25].

El principal motivo de la elección de Electron para el desarrollo de la herramienta se debe a que se pueden desarrolloar aplicaciones multiplataforma y ofrece la posibilidad de realizar llamadas a procedimientos remotos (RPC). Para la implementación de las llamadas a procedimientos remotos se ha utilizado el módulo ZeroRPC[2]. Con la integración de ZeroRPC se consigue la comunicación con los diferentes procesos implementados en otras tecnologías que se citan en los siguientes apartados.

Node.js

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Utiliza un modelo de operaciones entrada/salida sin bloqueos y orientado a objetos. Los paquetes "npm" de Node.js son el ecosistemas de librerías de código abierto más grande en la actualidad. [26]

Funciona con un modelo de evaluación de un único hilo de ejecución utilizando entradas y salidas asíncronas, las cuales pueden ejecutarse concurrentemente. La arquitectura de un único hilo de ejecución para atender todas las solicitudes hace que todas las operaciones que realicen entradas y salidas implementen una función Callback [26]. Incorpora módulos básicos como el módulo de red que proporciona una capa para programación asíncrona u otros módulos como Path, File System o "fs", Buffer. Existe la posibilidad de añadir módulos de terceros.

Con la elección de Electron es necesario la integración de Node.js en la arquitectura del proyecto. Una gran ventaja es la cantidad de paquetes desarrollados por terceros que facilitan y simplifican la implementación de la herramienta [26].

JQuery

Se trata de una biblioteca multiplataforma de JavaScript, que facilita la interacción con documentos HTML, manipular el árbol DOM[3], manejar eventos y desarrollar animaciones.

La herramienta que se implementa en el proyecto sigue una metodología orientada a eventos. Se elige *JQuery* debido a que esta tecnología permite capturar eventos de los componentes HTML y facilita de alguna forma la ejecución de las funcionalidades que se implementan en Node.js.

Python

Python es un lenguaje de programación multiparadigma soportando la orientación a objetos, programación imperativa y programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma [28].

Puede distribuirse libremente ya que su intérprete como la biblioteca estándar se encuentra de forma binaria y en código fuente en la página web de Python. En la web se pueden encontrar también distribuciones y módulos desarrollados por terceros, así como documentación adicional. El intérprete de Python puede extenderse fácilmente con nuevas funcionalidades y tipos de datos implementado en C o C++ [28].

[2]Es una biblioteca para la comunicación distribuida entre procesos de la parte del servidor basada en ZeroMQ y MessagePack.

[3]Sistema de herramientas implementadas para la manipulación de archivos XML.

Es un lenguaje que está ganando usuarios que se dedican al campo de la inteligencia artificial. El *framework* Keras utilizado para el entrenamiento de redes neuronales es una *API* integrada en Python, siendo esta la razón de la elección para la integración en el proyecto. Mediante llamadas a procesos remotos se puede establecer la comunicación con la herramienta desarrollada con Electron.

ZMQ

En ocasiones es necesario conectar diferentes partes de un software para establecer una comunicación entre ellas. El problema surge cuando las dos partes que se quieren conectar están implementadas en lenguajes diferentes.

ZeroMQ es un sistema de mensajería con la capacidad de solucionar el problema mencionado en el párrafo anterior. Es capaz de conectar códigos en casi cualquier lenguaje y cualquier plataforma. Los creadores argumentan que ofrece un gran entorno de concurrencia, resolviendo entre otros la comunicación entre hilos de un proceso o entre procesos de una misma máquina [29]. Ofrece una serie de patrones de diseño que solucionan problemas comunes, entre estos patrones se encuentran: *Req-Rep, Push-Pull, Router-Dealer o Dealer-Dealer* [30].

Durante la fase de entrenamiento, el proceso Python devuelve información relevante sobre el estado del entrenamiento. Esta información se debe actualizar en la interfaz de la herramienta. Con la integración de ZeroMQ se consigue la comunicación entre el proceso Python y el proceso Node.js encargado de actualizar la interfaz de usuario.

Materialize

Creado y diseñado por Google, es un lenguaje de diseño del interfaz de una aplicación que combina los principios clásicos del diseño con los más actuales. La finalidad de Google es ofrecer una experiencia de usuario única en todos sus productos en cualquier plataforma.

Materialize cuenta con componentes con estilos por defecto, pero a su vez fáciles de manejar y modificar por el usuario. Pensado principalmente para el desarrollo web pero se puede integrar en aplicaciones para dispositivos móviles como en aplicaciones para escritorio [31]. La hoja de estilos propia permite que cada diseño se pueda adaptar fácilmente a cualquier resolución de pantalla. En la documentación oficial se encuentran un gran número de ejemplos de cómo utilizar los componentes tradicionales y los más actuales.

La interfaz de usuario de el proyecto se genera a partir del HTML creado en el proyecto de Electron. Después de realizar pruebas con Materialize se ha comprobado la facilidad de aplicar un diseño atractivo para la interfaz, ofreciendo una experiencia de usuario positiva.

TensorFlow

TensorFlow es una biblioteca de software de código abierto para el aprendizaje automático. Fue creado y lanzado por Google en el año 2015 consiguiendo un gran éxito hasta el momento. Cuenta con una gran cantidad de colaboradores que lo sitúan como líder en el sector del aprendizaje profundo [32].

Se trata de una plataforma creada para la construcción y entrenamiento de redes neuronales, que permite detectar y descifrar patrones propios de los seres humanos [33].

Puede realizar operaciones de cálculo tanto en CPU o GPU. Para el entrenamiento de redes neuronales convolucionales ofrece un extenso soporte para trabajar con GPU tanto de equipos de escritorio como en servicios Cloud.

Para el presente proyecto se hace uso indirecto de Tensorflow debido a la integración de la biblioteca Keras. En el siguiente apartado se profundiza sobre el funcionamiento de Keras. En el Apéndice A se especifica la máquina en la que se ha realizado la instalación y en el Apéndice B se explican los pasos seguidos para la instalación de TensorFlow GPU y Keras.

Keras

Keras es una *API* de redes neuronales de alto nivel, escrita en Python y con posibilidad de integrarse con TensorFlow, CNTK[4] o Theano[5]. Su desarrollo se basó en crear una *API* con la que realizar un experimento sobre una red fuese una tarea sencilla y rápida.

La facilidad de uso es uno de los principales objetivos, ofreciendo una *API* consistente y simple que minimiza el número de acciones que ha de realizar el usuario. Ofrece una documentación extensa con algunos ejemplos que pueden ser de gran ayuda para empezar a trabajar en el campo de las redes neuronales. Separa por módulos cada una de las partes de una red neuronal. Las capas de una red neuronal, la función de coste, la función de activación, la función de regularización son entre otros los módulos por los que está formado. Se pueden crear módulos personalizados fácilmente, haciendo de Keras un sistema apto para investigaciones avanzadas [34].

Los modelos se escriben en código Python. La implementación en Python ofrecen la ventaja a la hora de la depuración y extensión de los modelos. Gracias al desarrollo de librerías como Keras, Python está aumentando en el sentido de desarrolladores que trabajan con él.

Se ha elegido Keras para integrarlo en el proyecto por la facilidad de implementar una red neuronal. Existe una gran comunidad de desarrolladores que comparten numerosos ejemplos de código que, en cierta manera, ayudan al desarrollo de la herramienta.

OpenCV

OpenCV es una biblioteca libre de visión por computador desarrollada por Intel. Desde su aparición en 1999 ha sido utilizado en infinidad de aplicaciones. Esto se debe a que su publicación se da bajo licencia *Berkeley Software Distribution* (BSD), que permite que sea usado libremente para fines comerciales y de investigación con las condiciones expresadas.[35]

Es multiplataforma, con versiones para distribuciones GNU/Linux, Mac OS X, Windows, Android e IOS. Tiene interfaces C ++, Python y Java.Sus funciones abarcan grandes áreas en el proceso de visión como el reconocimiento de objetos, calibración de cámaras o visión robótica.[35]

Otras funciones como redimensionamiento o recorte de imágenes también están incluidas en la biblioteca. Estas características junto la detección de objetos han decantado la elección para la integración en el proyecto.

[4]Es un framework basado en aprendizaje profundo desarrollado por Microsoft Research.

[5]Es un proyecto open source desarrollado por el grupo de Machine Learning de la Universidad de Montreal.

3.4 Implementación

En la presente herramienta el proceso principal es el encargado de inicializar el proceso renderizador, encargado de mostrar la interfaz gráfica. Por otra parte el proceso principal también se encarga de inicializar un proceso Python si este no ha sido inicializado por el usuario de forma manual.

El proceso Python inicializado es el encargado de ejecutar todas la tareas relacionadas con el *framework* Keras y la librería OpenCV. Este proceso se trata básicamente de una *API* donde el usuario realiza una petición a una tarea en concreto y el servidor es el encargado de atender estas peticiones. De aquí en adelante se nombrará a este fichero como servidor Python.

Para realizar la comunicación entre el cliente y el servidor se ha hecho uso de la tecnología ZeroRPC. El cliente de la aplicación es el proceso renderizador y el servidor es el proceso Python (ver Figugra 3.2). Al inicializar la herramienta el servidor Python se pone a la escucha para atender las peticiones del usuario. Cuando el usuario interactúa con la interfaz gráfica hay diferentes funciones en la parte del cliente que se encuentran a la escucha para lanzar la correspondiente petición al servidor.

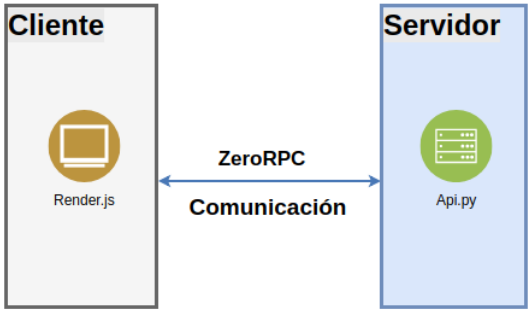

Figura 3.2: Comunicación entre procesos cliente y servidor.

En los siguientes apartados se describen todas las comunicaciones que se han implementado. Algunas de las peticiones lanzan un proceso en el servidor Python que generan más de una respuesta. Para este tipo de peticiones se ha implementado una arquitectura adicional que captura cada uno de los mensajes y los envía para que puedan ser mostrados en la interfaz gráfica.

3.5 Funcionalidad

En los siguientes apartados se describe la funcionalidad de cada una de las partes de la herramienta implementada para el entrenamiento de redes convolucionales. Se exponen las funcionalidades de cada una de las partes y los detalles de implementación de estas especificando cuáles de las tecnologías explicadas anteriormente se han utilizado.

Las secciones se presentan de forma ordenada para entender el correcto funcionamiento de la herramienta. Para acceder a cada una de las secciones se ha implementado un menú lateral (ver Figura 3.3) en la herramienta que siempre está accesible para cambiar de sección.

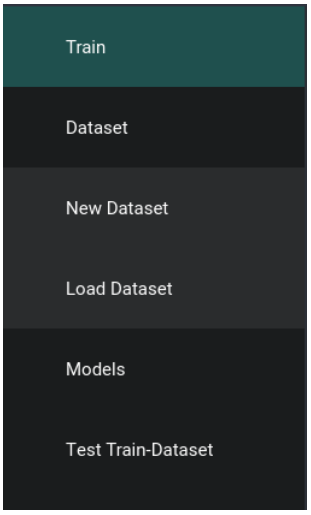

Figura 3.3: Menú principal de la herramienta.

3.5.1. Crear conjunto de imágenes

Descripción

Selección de directorios y número de clases

Un conjunto de imágenes está formado por las imágenes de entrenamiento e imágenes de validación. Este conjunto de imágenes se ha de estructurar de forma correcta para poder realizar el entrenamiento de la red. El conjunto de imágenes se organiza en directorios de entrenamiento y validación y cada uno de ellos en directorios con las correspondientes clases de las imágenes (ver Figura 3.4).

El usuario tiene una opción en el menú principal llamada *Data Set* donde puede elegir entre *New Data Set* o *Load Data Set*. En esta sección se explica la opción *New Data Set* cuya finalidad es la creación de un conjunto de datos para el posterior entrenamiento.

```
--./directory_images
    --./train
        --./class_1
            .img1
            .img2
            ...
        --./class_2
            ...
    --./validation
        --./class_1
            .img1
            .img2
            ...
        --./class_2
            ...
```

Figura 3.4: Estructura directorio de imágenes.

El principal objetivo de este apartado es facilitar al usuario la creación de un conjunto de imágenes de forma estructurada. En primer lugar el usuario selecciona los directorios donde se quieren almacenar las imágenes de entrenamiento y de validación. A continuación debe indicar el número de clases por las que está formado su conjunto de imágenes.

Para la selección de directorios se abre una nueva ventana con los directorios de la máquina del usuario, donde puede navegar y seleccionar el directorio deseado. Una vez seleccionados los directorio la herramienta crea los directorios de *Train* y *Validation* en la ruta seleccionada. Seleccionados los directorios de validación y entrenamiento, el usuario

indica el número de clases. Se crean tantas carpetas en los directorios de entrenamiento y validación como clases se han indicado. No se podrán indicar el número de clases hasta que no se indiquen los directorios de entrenamiento y validación.

Añadir, eliminar y editar imágenes de una clase

Una vez se han definidos los directorios y el número de clases, se crea en la interfaz gráfica un sistema de pestañas donde cada pestaña se corresponde con una clase (ver Figura 3.5). Este sistema de pestañas se genera de forma dinámica dependiendo el número de clases que se hayan especificado.

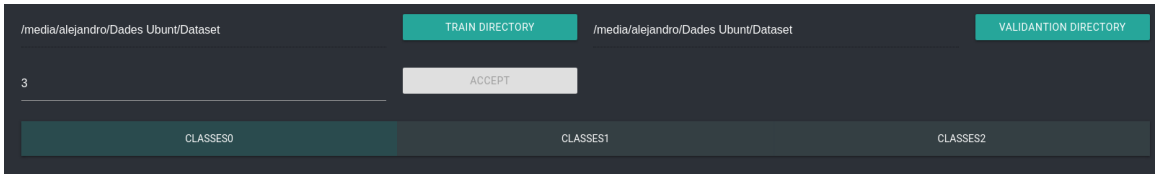

Figura 3.5: Sistema de pestañas.

Todas las pestañas están compuestas por los mismas opciones para cada clase. Se puede establecer un nombre a cada una de las clases, en caso contrario se establece por defecto. El usuario tiene la opción de añadir imágenes de entrenamiento y de validación así como eliminar cualquiera de las que haya introducido. Para añadir o eliminar se muestra un ventana con el sistema de ficheros donde se pueden seleccionar una o más imágenes simultáneamente.

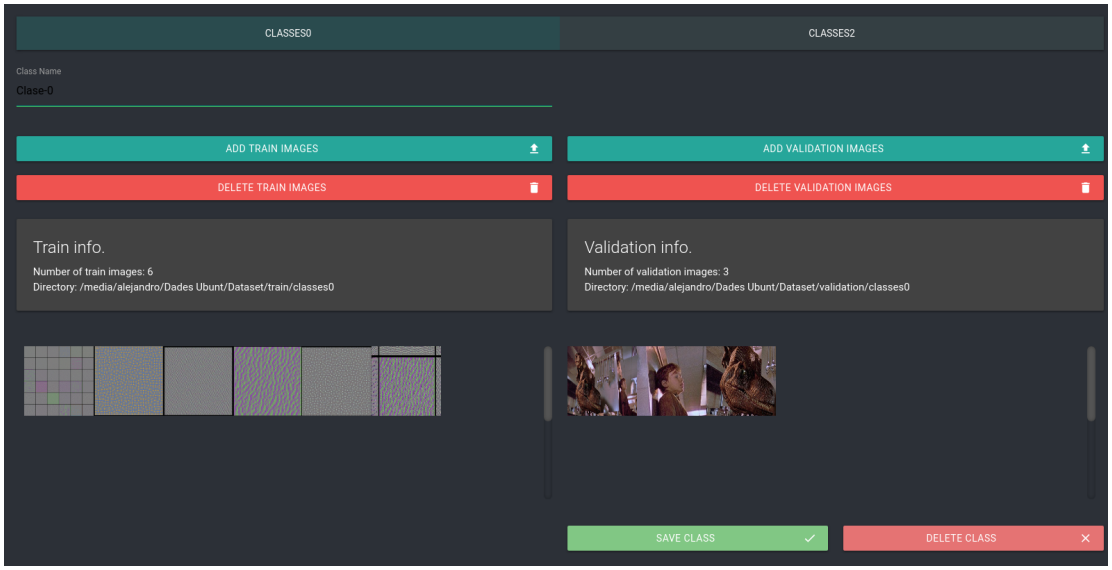

Figura 3.6: Opciones del sistema de pestañas.

Cada una de las pestañas está dividida en dos columnas para separar las opciones de entrenamiento y validación y ofrecer una mejor experiencia de usuario (ver Figura 3.6). Por cada conjunto de imágenes introducido se actualizan dos paneles con información básica de la clase y una galería que permite visualizar todas las imágenes de dicha clase. Cuando el usuario accede a una de las imágenes de la galería, puede eliminar o modificar esta. La funcionalidad de edición permite al usuario recortar la imagen por diferentes partes de esta y añadir estos recortes como nuevas imágenes en el conjunto de datos.

Al finalizar la creación de una clase, el usuario tiene la opción de guardar la configuración de esta. Posteriormente tiene la posibilidad de volver a la pestaña de la clase

guardada y modificar el contenido y volver a guardar la nueva configuración. Otra opción es la eliminación de la clase, eliminando todo el contenido de esta.

Guardar conjunto de datos

Para completar esta pantalla el usuario tiene la opción de guardar el conjunto de imágenes o eliminarlo completamente. Si la opción elegida es guardar el conjunto de imágenes, el usuario debe añadir un nombre a este conjunto y seleccionar la ubicación donde desea guardarlo. Se genera un fichero JSON con toda la información necesaria para posteriormente poder cargarlo y trabajar con este. En los próximos subapartados se describe el contenido de este fichero.

Implementación

Selección de directorios

La creación de un conjunto de imágenes necesita mantener una estructura con los datos que se van almacenando para poder acceder a ellos en cualquier momento de forma correcta. La selección de un directorio para almacenar las imágenes de entrenamiento o de validación ha de quedar guardada para posteriormente saber donde ubicar las imágenes que el usuario añada a este conjunto. El módulo "fs" provee una *API* para interactuar con el sistema de ficheros del sistema. Al seleccionar un directorio se crean las carpetas de entrenamiento o validación en el directorio especificado.

Añadir, eliminar y editar imágenes de una clase

La principal funcionalidad de esta pantalla es añadir imágenes en cada una de las clases que el usuario haya seleccionado. Para crear las clases el usuario establece un determinado número y cuando presiona sobre el botón de "Aceptar" estas clases son creadas y aparecen en un sistema de pestañas. De forma dinámica es posible añadir pestañas, pero la complejidad está en tratar a cada una de forma individual. Para ello se añade un identificador diferente en los componentes HTML que conforman una pestaña. Todas las clases tienen un identificador único que es almacenado en la estructura JSON. Para saber con qué clase se está trabajando cada momento, se implementa un evento que captura el identificador de la pestaña. El sistema de pestañas, a partir de funciones implementadas en Node.js la herramienta permite:

- **Contar el número de imágenes de validación y entrenamiento que se encuentran en el directorio**. Se implementa una función que almacena el nombre de todos los ficheros de un directorio y después se hace el recuento de elementos que forman la lista. En la estructura JSON además del número de imágenes por cada clase también se almacena el número total de imágenes.

- **Añadir y eliminar imágenes**. Para añadir una imagen se implementa una ventana de ficheros donde el usuario selecciona las imágenes que quieres añadir. Por cada imagen seleccionada se almacena el directorio y el nombre de esta para copiarlas en el directorio de la clase correspondiente. La acción de eliminar sigue el mismo procedimiento pero en este caso implementa la función para eliminar archivos con el módulo "fs".

- **Cargar, de forma dinámica, todas las imágenes para que se puedan visualizar en la galería**. La galería se modifica para añadir dos opciones cuando se requiera la visualización de una imagen:

- Editar imagen: Se ha añadido un nuevo icono a modo de botón que, cuando es presionado por el usuario, salta un evento. Con la finalidad de lanzar una petición al servidor Python para ejecutar un nuevo proceso OpenCV, encargado de facilitar los recortes en la imagen. En la llamada se pasa como argumento la ruta donde se encuentra la imagen.

- Eliminar imagen: Esta funcionalidad también se implementa fuera de la galería para visualizar qué imagen se está eliminando. Se establece un evento que cuando el usuario presiona sobre el icono de borrado el evento captura la acción y, con el módulo "fs" eliminamos la imagen.

Para la galería se utiliza un módulo específico para Node.js llamado *Light Gallery*.

- **Guardar o eliminar una clase**. Una clase puede ser guardada cuando se modifica algún dato, en el resto de los casos el botón se encuentra inhabilitado. Por otra parte, se encuentra la opción de eliminar la clase, que si es seleccionada se elimina la pestaña actual, los directorio e imágenes de la clase y el objeto JSON.

Guardar conjunto de datos

La opción de guardar o eliminar el conjunto de imágenes son las opciones que completan el módulo. La primera opción tiene la funcionalidad de crear un fichero de tipo JSON con toda la información que se ha almacenando. Este fichero es necesario para el usuario para posteriormente cargar el conjunto de datos para tareas de entrenamiento o realizar alguna modificación en los datos de este. La opción de eliminar el conjunto de datos restaura todos los valores del módulo para que el usuario empiece a crear un nuevo conjunto desde cero.

Estructura fichero JSON

Tal y como se ha mencionado en el apartado anterior, cuando se guarda la configuración de un conjunto de imagen se almacena en un fichero JSON (ver Figura 3.7). Los datos guardados son necesarios para posteriormente cargar el conjunto de imágenes y realizar modificaciones o bien utilizarlo en el entrenamiento de una red. Los datos almacenados se pueden dividir en datos generales del conjunto de imágenes y datos individuales de cada una de las clases. En los datos generales se recoge, el nombre del conjunto, directorios de entrenamiento y validación, el número de clases y el tamaño de las imágenes. En relación a las clases, se almacena el nombre de la clase, el directorio de la imágenes de entrenamiento y validación y el número de imágenes en cada uno de estos directorios.

3.5.2. Cargar conjunto de datos

Descripción

En apartados anteriores se explica cómo crear un conjunto de datos y almacenar la información de este. En esta pantalla se accede a la información que se guardó al crear un conjunto de datos. La funcionalidad es prácticamente igual a la que se ha descrito en la sección de crear un conjunto de datos. El usuario debe seleccionar el fichero guardado con la información del conjunto de datos y a continuación se genera un sistema de pestañas exactamente igual al de la pantalla para crear un nuevo conjunto de datos.

```
{
   "name":"dataset-A",
   "train_dir":"/media/alejandro/Dades Ubunt/Dataset/train",
   "validation_dir":"/media/alejandro/Dades Ubunt/Dataset/validation",
   "image_size":"500*271",
   "classes":{
      "num_classes":2,
      "info":{
         "classes0":{
            "name_class":"Gossos",
            "train_dir":"/media/alejandro/Dades Ubunt/Dataset/train/Gossos",
            "validation_dir":"/media/alejandro/Dades Ubunt/Dataset/validation/Gossos",
            "num_train":6,
            "num_validation":3
         },
         "classes2":{
            "name_class":"Gats",
            "train_dir":"/media/alejandro/Dades Ubunt/Dataset/train/Gats",
            "validation_dir":"/media/alejandro/Dades Ubunt/Dataset/validation/Gats",
            "num_train":9,
            "num_validation":5
         }
      }
   }
}
```


Figura 3.7: Estructura fichero JSON.

Implementación

La carga de un conjunto de datos se realiza a partir del archivo creado cuando se guarda un conjunto en su pantalla correspondiente. La funcionalidad es exactamente igual a la definida en el apartado anterior a diferencia de la opción de carga de archivos.

La dificultad de esta pantalla se encuentra en cargar correctamente el fichero JSON y tratar estos datos de forma correcta para poder generar dinámicamente todos los componentes del sistema de pestañas. Cada componente tiene un evento que se encuentra a la escucha de una posible acción del usuario, y cada componente tiene un identificador para poder ejecutar una acción cuando se interactúe con él. El hecho de tener dos pantallas con una parte exactamente igual no significa que estás compartan el mismo código ya que cada una debe tener unos identificadores propios para atender a cada una de las acciones. Una de las ventajas es que el usuario puede interactuar con estas dos pantallas simultáneamente.

3.5.3. Creación de una red neuronal convolucional

Descripción

Toda red neuronal tiene definido un modelo o, llamado de otra forma, una arquitectura. Actualmente existen diferentes arquitecturas incluidas en la *API* de Keras que se pueden cargar y utilizar como modelos preentrenados. Las arquitecturas actuales disponibles en Keras se presentaron en la Tabla 2.1 donde se exponen las principales características de los modelos especificando, entre otros, la profundidad en número de capas de los modelos.

El usuario dispone de dos opciones en esta pantalla, crear un modelo a partir de una arquitectura disponible en la *API* de Keras o la creación de una arquitectura personaliza-

da.

Arquitectura de la *API* Keras

La configuración de esta pantalla basada, en modelos preentrenadoss, tiene un enfoque a la transferencia de aprendizaje.Cuando el usuario seleccione la opción de trabajar con modelos preentrenados, aparece un desplegable (ver Figura 3.8) donde seleccionar una de las redes mencionadas en la Tabla 2.1.

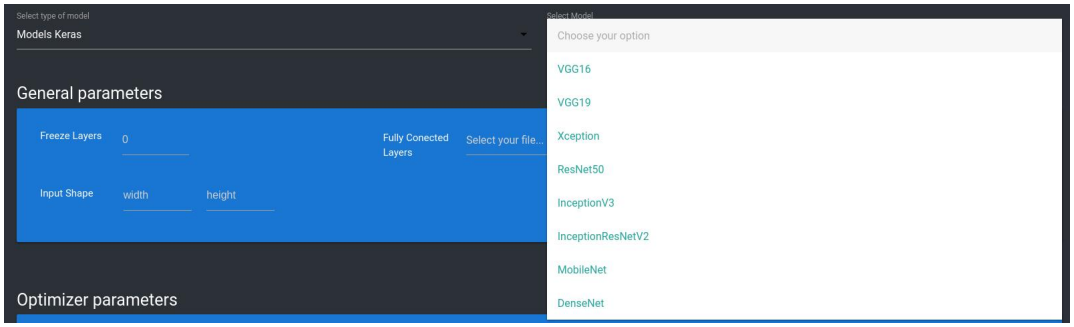

Figura 3.8: Desplegable con los modelos preentrenados disponibles.

Para la utilización de una red preentrenada el usuario debe añadir las últimas capas totalmente conectadas para obtener un resultado acorde con el de su conjunto de datos. Si el usuario quiere realizar un entrenamiento para un número determinado de clases, debe especificarlo en la salida de la red. Esto significa que no todos los modelos creados serán aptos para cualquier conjunto de datos.

Sobre los modelos anteriores el usuario también tiene la opción de habilitar algunas capas para no utilizar los valores cargados para este y entrenarlos con el conjunto de datos elegido. Cuantas más capas se habiliten para ser entrenadas más tiempo llevará el proceso de entrenamiento.

En cada modelo se ha de establecer un tamaño de entrada que se corresponde con el ancho, alto y número de canales de una imagen (ver Figura 3.9). La entrada definida por el usuario para un modelo específico debe ser el que ha establecido en su conjunto de datos. Si se establecen unos valores incorrectos la red devuelve un error y no se realizará el entrenamiento.

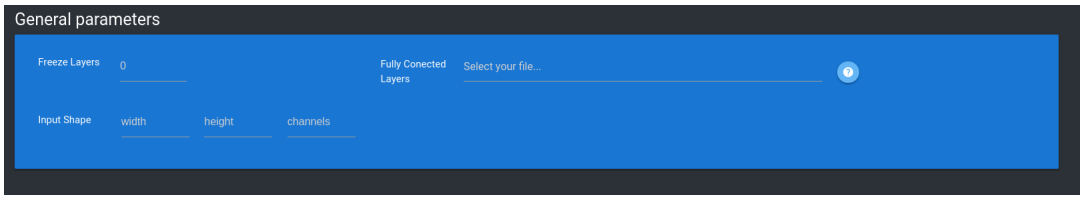

Figura 3.9: Parámetros generales de un modelo.

La compilación de un modelo requiere de dos parámetros necesarios a definir, un optimizador y una función de pérdida. A continuación se listan los optimizadores disponibles y sus parámetros accesibles en Keras:

- **SGD**. Parámetros: *learning rate* o lr, momentum, decay y nesterov.
- **RMSprop**. Parámetros: lr, rho, epsilon y decay.
- **Adagrad**. Parámetros: lr, rho, epsilon y decay.

- **Adadelta**. Parámetros: lr, epsilon y decay.
- **Adam**. Parámetros: lr, beta1, beta2, epsilon, decay y amsgrad.
- **Adamax**. Parámetros: lr, beta1, beta2, epsilon y decay.
- **Nadam**. Parámetros: lr, beta1, beta2 y epsilon.

Al seleccionar un optimizador la herramienta carga de forma dinámica los parámetros que se encuentran almacenados en un fichero JSON que se ha creado e insertado en la estructura del proyecto. En la página oficial de Keras se recomienda no alterar el valor de algunos parámetros. Para ofrecer más facilidad y ayudar al usuario, se añade un botón de ayuda para cada optimizador dónde se explica brevemente la funcionalidad del optimizador, requisitos y consejos para utilizar los parámetros.

La función de error es el segundo parámetro requerido para poder compilar un modelo. Se han incluido un total de 14 funciones, las cuales se encuentran disponibles en la *API* de Keras. En esta opción también se ha añade un botón de ayuda con información sobre estas funciones.

Finalizada la configuración de una de las redes preentrenadas, el usuario tiene la opción de guardar su modelo para ser utilizado posteriormente.

Arquitectura propia

En este apartado de la aplicación el usuario dispone de la opción de cargar un archivo con el modelo de red. También dispone de un botón de ayuda a modo de guía de cómo debe ser la estructura del fichero donde se encuentra el modelo definido.

El usuario debe importar un archivo con el modelo de red completo. En primer lugar se especifica la entrada de la red, que coincidirá con las dimensiones de las imágenes de su conjunto de datos, explicado en el apartado anterior. A continuación añadirá las capas convolucionales y *pooling* de la red, así como las capas totalmente conectadas que también deben ir acorde al número de clases del conjunto de datos. Este modelo se almacena en un directorio específico donde se encuentran todos los modelos guardados, como los archivos de los modelos preentrenados.

Para finalizar la configuración se deben especificar los parámetro requeridos para compilar el modelo, un optimizador y una función de error. Estas opciones son exactamente las mismas que se explican en el apartado anterior.

Como en el anterior apartado, si el usuario desea guardar el modelo creado se creará un fichero para poder cargar este modelo a la hora de realizar un entrenamiento.

Implementación

Gracias a la modularidad que ofrece Keras para el desarrollo de un modelo, ha sido posible implementar esta pantalla donde el usuario puede crear una red por partes para posteriormente importar el modelo completo al fichero que ejecuta el entrenamiento. A continuación se explica con detalles las opciones disponibles para crear un modelo nuevo.

Se pueden incluir dos tipos de modelos, uno con una red preentrenada o un modelo implementado por el usuario. En la parte superior de la pantalla se sitúa un componente 'select' con las dos opciones donde, para cada una de estas dos opciones se genera información diferente para la visualización e interacción del usuario. La separación de este tipo de contenido se realiza a partir de dos componentes 'divs', y en cada uno de ellos,

se cargan los componentes específicos de cada modelo. Cada una de las opciones tienen un identificador propio, para así poder lanzar un evento personalizado para cargar unas opciones u otras.

Arquitectura de la *API* Keras

Si se opta por la opción de cargar un modelo de red preentrenado, se cargan tres paneles con diferentes campos a completar por el usuario. En el primer panel existen unos componentes simples que se han mencionado en la funcionalidad de la herramienta. En este primer panel es importante almacenar el nombre del fichero que añade el usuario, y este se almacena en el directorio donde se encuentran los diferentes modelos de red disponibles o creados con la herramienta . El fichero introducido por el usuario se corresponde con las últimas capas conectadas de un modelo. En el fichero Python encargado de realizar el entrenamiento, se importa el modelo seleccionado por el usuario a partir de el nombre de modelo de red preentrenada seleccionado. Con el nombre de la red, se busca un fichero con el mismo nombre en el directorio donde se encuentran los modelos.

El segundo panel implementado para la selección de un optimizador para la red, se accede a un fichero JSON donde se encuentran los optimizadores disponibles. Estos optimizadores son cargados en un componente 'select'. Cada optimizador tiene un conjunto de parámetros que el usuario puede completar o dejar a un valor por defecto. Si un optimizador es seleccionado se genera de forma dinámica cada uno de los parámetros del optimizador, los cuales están almacenados en el fichero JSON con sus valores por defecto. Cada optimizador tiene un identificador diferente y se almacena en una variable, con el fin de llevar un control de que optimizador ha sido seleccionado por el usuario. Si el modelo es guardado por el usuario, se almacena en el fichero JSON que optimizador ha sido seleccionado y los valores de sus parámetros. A partir de esta información, cuando se ejecuta un entrenamiento estos datos son pasados por argumentos al servidor Python que se encarga de establecer estos parámetros antes de compilar un modelo de red.

En el último panel se encuentra un componente 'select' con las funciones de error disponibles en la *API* de Keras. Si el usuario selecciona guardar el modelo debe añadir un nombre a este y seleccionar el directorio donde quiera almacenarlo.

Arquitectura propia

Esta parte también está conformada de tres paneles. El primer panel es el único que añade una funcionalidad diferente, donde el usuario debe elegir un fichero que contenga el modelo de una red completo. Cuando el fichero es seleccionado es copiado al directorio de modelos y se almacena la información sobre este en el archivo JSON que se genera al guardar el modelo.

El resto de paneles correspondientes al optimizador y a la función de error están implementados igual que se explica en el subapartado anterior.

Guardar modelo

Cuando el usuario ha creado correctamente el modelo tiene la opción de guardar los datos en un fichero JSON con todos los valores de la red para ejecutar posteriormente un entrenamiento. Dependiendo si el modelo de red creado es preentrenado o personalizado se guarda un valor en el fichero, para importar unos ficheros u otros a la hora de realizar el entrenamiento. El resto de valores se corresponden con los valores que se han completado en los diferentes paneles. En el próximo apartado se detalla cómo los valores almacenados son utilizados en los diferentes ficheros implementados para el entrenamiento.

3.5.4. Entrenamiento y monitorización de una red convolucional

Descripción

Está es posiblemente la pantalla más importante de la herramienta que se desarrolla. El objetivo de esta pantalla consiste en que el usuario pueda seleccionar un modelo de red y un conjunto de datos para realizar un entrenamiento. Durante el entrenamiento se visualizan los resultados que está ofreciendo el entrenamiento de la red seleccionada para un conjunto de datos.

Configuración inicial

Antes de realizar un entrenamiento se deben completar diferentes campos necesarios para poder lanzar el proceso de entrenamiento (ver Figura 3.10). En primer lugar el usuario debe seleccionar un modelo para entrenar su red, siendo uno de estos un modelo creado con la herramienta en la sección dedicada a la creación de modelos. Una vez seleccionado el modelo se debe seleccionar el conjunto de datos con el que se desea realizar el entrenamiento. El usuario selecciona el número de *epochs* y el tamaño de *batch size* donde:

- El número de *epochs* se corresponde con el número de iteraciones que ha de realizar el modelo de red para el entrenamiento.
- *Batch size* es un lote formado por un número de imágenes, si se tienen 100 imágenes y se selecciona un *batch size* igual a 50 la red entrenará primero con un lote de las imágenes de la 0 a 49 y posteriormente de 50 a 99. Para este parámetro se ha de tener en cuenta la capacidad de cálculo de la máquina en la que se realiza el entrenamiento, ya que si se establece un número elevado es posible que la máquina no pueda procesarlo.

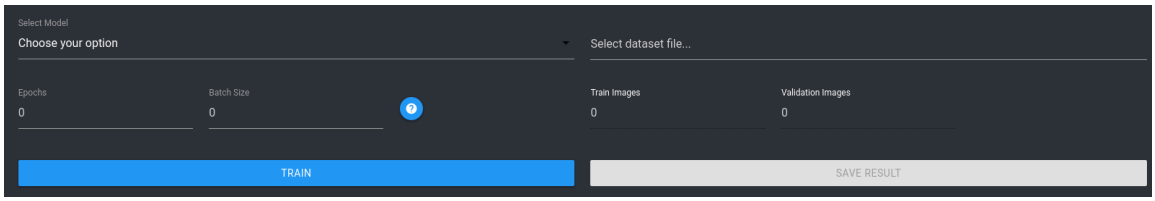

Figura 3.10: Configuración inicial para realizar un entrenamiento.

Monitorización de los resultados

Durante cada iteración realizada en el entrenamiento, la herramienta recibe estos resultados en tiempo real. En un primer panel se lleva la monitorización del tiempo transcurrido en el entrenamiento y el número de iteración en el que se encuentra el entrenamiento de la red. Estos resultados también son plasmados en dos gráficas donde, la primera gráfica está destinada a los resultados que devuelve la red para el conjunto de entrenamiento (ver Figura 3.11). Se puede visualizar la tasa de acierto y de error para este conjunto. El usuario puede situarse en un punto de la gráfica para consultar el resultado de forma rápida y sencilla. La segunda gráfica tiene la misma funcionalidad que esta primera a diferencia que los resultados se corresponden con el conjunto de datos de validación.

Guardar resultados

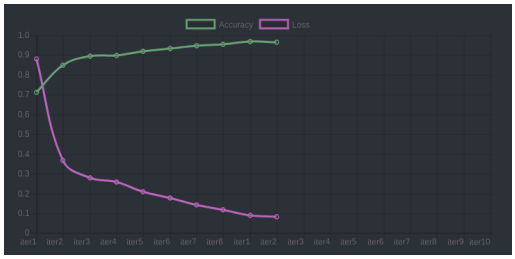

Figura 3.11: Gráfica resultado imágenes de entrenamiento.

Una vez finalizado el entrenamiento, se notifica al usuario con una ventana modal la finalización de este. El usuario tiene la opción de guardar los resultados, que se almacenan en un archivo JSON con los resultados del entrenamiento. Además el fichero JSON contiene la ruta a un archivo generado en formato "h5" por el *framework Keras* que será necesario para posteriormente realizar las pruebas *test*. La importancia de guardar este archivo, está en realizar pruebas con la red entrenada o simplemente para la visualización de los resultados que se obtuvieron durante el entrenamiento. En el subapartado "Resultados y pruebas de una red entrenada" se explican las opciones disponibles para el usuario a partir de un modelo de red entrenado.

Implementación

Configuración inicial

Como se menciona en la descripción de esta pantalla, para la configuración inicial se han de completar una serie de campos. El primer campo el usuario selecciona un modelo de red que previamente ha creado con la herramienta. Este archivo tiene un formato JSON y contiene todos los datos del modelo, los cuales se almacenan en una estructura para posteriormente ser enviados como parámetros al servidor Python. Como se explicó en el apartado de "Creación de una red neuronal convolucional", Keras ofrece modularidad la cual se aprovecha para la implementación de este apartado. Los valores pasados como parámetros al servidor Python son utilizados en los diferentes módulos creados con Keras. Para entender un poco mejor la distribución de estos parámetros se presenta el un esquema en la Figura 3.12.

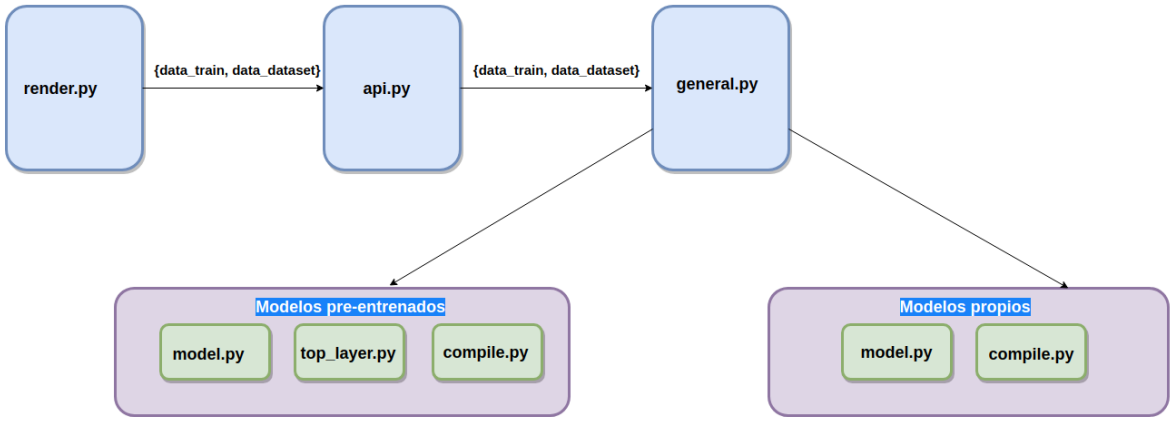

Figura 3.12: Esquema acción entrenamiento.

El usuario debe introducir también un conjunto de datos, el cual mantiene a su vez una estructura JSON. Algunos de los valores del conjunto de datos son pasados como parámetros así como los campos de *epochs* y *batch size*. Como se puede visualizar en la

Figura 3.12, todos los datos recogidos en la interfaz llegan al servidor Python (api.py) y después son pasados como argumentos hasta el fichero principal (general.py) que se encarga de realizar el proceso de entrenamiento de un modelo de red. El fichero que se encarga de realizar el entrenamiento actúa de una manera u otra dependiendo si se trata de un modelo preentrenado o personalizado del usuario. Las diferencias entre ambos son:

- Modelo preentrenado:
  - El modelo de red a entrenar es pasado por argumentos. A partir del nombre del modelo se importa el fichero correspondiente al modelo.
  - El nombre del fichero de las capas de salida es pasado como argumento para ser importado en el programa principal.
  - Se importa el fichero encargado de compilar el modelo. Antes de lanzar el entrenamiento se llama a la función encargada de compilar el modelo, la cual recibe como parámetros el optimizador de red y sus parámetros, la función de error y el número de capas que se pueden entrenar.
- Modelo propio:
  - El modelo de red a entrenar es pasado por argumentos. A partir del nombre del modelo se importa el fichero correspondiente al modelo.
  - Se importa el fichero encargado de compilar el modelo. Antes de lanzar el entrenamiento se llama a la función para compilar el modelo, a esta función se le pasa el optimizador de red y sus parámetros y la función de error.

Debido a la complejidad de explicar este apartado para que el lector pueda entender cómo funciona esta parte de la herramienta, se ha optado por añadir el código utilizado y la estructura de directorios en el Apéndice D con la finalidad de facilitar una ayuda y mostrar el trabajo realizado.

Monitorización de los resultados

Una vez completados correctamente todos los parámetros, se habilita la opción para poder ejecutar el entrenamiento de la red. Al ejecutar la opción de entrenamiento se realiza una comunicación entre el cliente y el servidor Python. Como se menciona en el apartado anterior, se implementa un *script* Python que se encarga de realizar el entrenamiento haciendo uso del *framework* Keras.

Además de los módulos explicados anteriormente para realizar el entrenamiento, se implementa un módulo de *Callbacks* en Python, encargado de devolver los valores de acierto y error al final de cada iteración. Estos últimos valores son los que se muestra en la interfaz gráfica para que el usuario pueda monitorizar el estado y resultado del entrenamiento. En este módulo se ha implementado una arquitectura que se explica más adelante para la transmisión de estos valores hasta la interfaz.

Para monitorizar el estado del entrenamiento, se ha implementado un panel en la interfaz gráfica que indica al usuario el tiempo transcurrido y la iteración en la que se encuentra el entrenamiento. Seguido de este panel se añaden dos gráficas a partir de un módulo de *Node* llamado *Chart*. La razón de la elección de este módulo específico se debe a la cantidad de gráficos disponibles y a la facilidad de integrarlo a la herramienta. Los resultados se visualizan en dos gráficos multilínea, concretamente dos líneas por gráfico, donde las líneas representan el acierto y el error respectivamente.

Se implementa una arquitectura adicional con la tecnología ZeroMQ para recibir los resultados al final de cada iteración . Dos *routers* se encargan de atender las peticiones del

cliente (interfaz) y del *worker* (*Callbacks* Keras). El cliente de la interfaz tiene un identificador único que es pasado en los argumentos cuando se lanza una tarea de entrenamiento. Al inicializar la herramienta el cliente envía un mensaje al *router*_1 para indicar su disponibilidad y quedar a la espera de los mensajes de la parte del *router*_2. En la parte del *worker*, este recibe unos valores del entrenamiento al final de cada iteración y es el encargado de enviarlos al *router*_2. Para realizar la comunicación de forma efectiva el *router*_2 envía un conjunto de datos en el que se incluye el identificador de el cliente y los datos del entrenamiento. El identificador del cliente es necesario ya que el *router*_2 debe redirigir el mensaje al cliente (interfaz) con este identificador. Para entender la arquitectura ver la Figura 3.13.

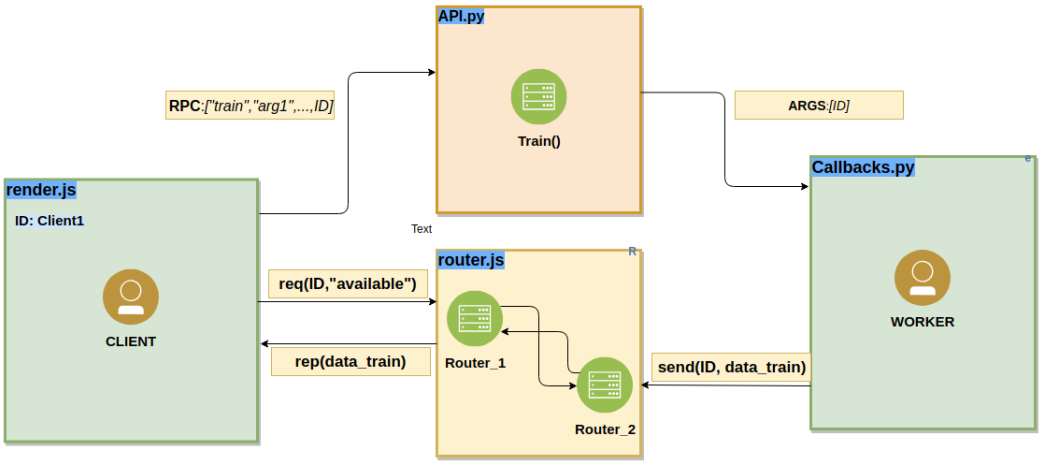

Figura 3.13: Comunicación entre *worker* (Keras) y cliente (interfaz).

3.5.5. Pruebas sobre una red entrenada

Descripción

Se trata de la última pantalla de la herramienta donde el usuario puede examinar los resultados de una red entrenada. Para ver los resultados se ha de seleccionar el fichero JSON que se guardó al finalizar la tarea de entrenamiento. En este fichero se almacenan los valores obtenidos en cada una de las iteraciones y son mostrados al usuario (ver Figura 3.14). Otra opción disponible es la de seleccionar una imagen para comprobar si se clasifica en la clase correcta.

Figura 3.14: Resultados obtenidos durante un entrenamiento.

Al seleccionar una nueva imagen, las predicciones de las anteriores son borradas para ocupar el mínimo espacio en la interfaz. Si se carga el resultado de otra red los datos de la anterior también desaparecen de la interfaz para cargar los nuevos datos.

Implementación

Para mostrar los resultados generados en el entrenamiento de una red, el usuario ha de seleccionar el fichero que guardó cuando se completó un entrenamiento. En este fichero se encuentran los resultados de todas las iteraciones, los cuales se presentan en una tabla. La tabla se genera de forma dinámica y consta de tantas filas como iteraciones tuvo el entrenamiento.

La segunda opción que ofrece este módulo es seleccionar una imagen para realizar la clasificación de esta. Si el usuario selecciona una imagen se realiza una petición al servidor Python, donde se pasa por argumentos el nombre y directorio de la imagen y el nombre del fichero "h5". Este último fichero es generado por Keras cuando se finaliza un entrenamiento y es el encargado de cargar la red para probar resultados. El servidor Python se encarga de lanzar un nuevo proceso Python que predice un resultado para la imagen seleccionada y este resultado será devuelto hasta la interfaz para mostrar el resultado. Para devolver el resultado se ha implementado una comunicación con ZMQ similar a la mencionada anteriormente que se encarga de hacerlo llegar hasta la interfaz del cliente.

3.6 Usuarios

En este apartado se especifican los usuarios a los que está dirigida la herramienta. Para cada usuario se detallan los conocimientos básicos que han de poseer para poder utilizar la herramienta de forma correcta.

La herramienta está enfocada a usuarios principiantes en el mundo de las redes convolucionales que sean capaces de entender los parámetros requeridos en la creación de un modelo de red convolucional. Otro requisito para este tipo de usuarios es poseer conocimiento medios en el lenguaje de programación Python y del framework Keras para el modelado de redes neuronales.

Sin embargo, la herramienta también puede ser utilizada por usuarios expertos, aunque pueden encontrar limitaciones a la hora de crear un modelo personalizado. En la creación de un conjunto de datos no encontrará limitaciones y puede aprovechar la herramienta para administrar de forma visual su conjunto.

No se requieren conocimientos en el campo redes convolucionales si el usuario final de la aplicación solo tiene la intención de crear un conjunto de datos.

CAPÍTULO 4
Problema planteado para la detección y reconocimiento de objetos

En este capítulo se plantea un problema práctico, que utiliza la herramienta desarrollada, enfocado a la detección de objetos para el cual se aportan diferentes soluciones y resultados con la finalidad de extraer conclusiones sobre el uso de las técnicas de aprendizaje profundo y resolver el problema de la mejor forma posible.

La detección de objetos es una de las prácticas más habituales en el campo de la visión por computador. En este caso se plantea un problema a resolver haciendo uso de técnicas para la visión por computador y de la herramienta implementada para modelos de aprendizaje profundo.

4.1 Descripción

En la actualidad existe un programa de televisión llamado El Hormiguero[1] donde semanalmente se realizan concursos para la gente que está viendo el programa desde casa o para la gente que acude como público al plató. Cada semana se realiza un concurso diferente, no repitiendo siempre el mismo. Para la gente que sigue el programa desde casa, existen dos tipos de concurso:

- **Llamada aleatoria**: el programa llama a un número de teléfono aleatorio y realiza una pregunta al dueño de la línea. Si este contesta correctamente a la pregunta se convierte en ganador de un premio de, mínimo, 3000€.
- **Llamada más rápida**: este concurso es diferente al anterior, la gente desde casa es la que ha de realizar una llamada al programa. El programa establece una cuenta atrás de 30 segundos y cuando esta termina aparece un número de teléfono en la pantalla. El telespectador que marca antes este número de teléfono es el ganador del concurso, ganando así un premio de, mínimo, 3000€.

El problema que se plantea en este capítulo se basa en el último concurso mencionado. Durante la emisión de un programa se realizó este concurso y se decidió participar sin éxito. El tiempo medio para marcar el número de teléfono desde que aparece en la

[1] El hormiguero es un *talk show* de televisión producido por 7 y acción para la cadena española Cuatro hasta septiembre de 2011 y en Antena 3 desde septiembre de 2011.

pantalla de la televisión oscila entre 2 y 4 segundos. Además a estos segundos se ha de añadir el retraso de la emisión del programa dependiendo de la zona geográfica.

Una vez se terminó la emisión en la que se realizó el concurso se pensó en realizar una aplicación que se encargará de detectar el número de teléfono y realizará la llamada automáticamente con el objetivo de conseguir un tiempo inferior a 2 segundos. Es importante remarcar que el objetivo del problema planteado no es ganar el concurso, sino intentar resolver un problema, que tiene unas restricciones temporales, utilizando las técnicas de aprendizaje profundo para conseguir un tiempo de respuesta similar al que puede tardar una persona para resolverlo.

Para el desarrollo se ha dividido el problema en diferentes subproblemas a resolver. Estos se mencionan de forma ordenada:

- Creación de un conjunto de datos, que contenga imágenes de los número del 0-9.
- Entrenar una red convolucional con el conjunto de datos creado.
- Detectar objetos en una escena determinada.
- Reconocimiento de cada uno de los objetos extraídos.
- Marcación automática del número obtenido.

En este capítulo se han utilizado imágenes extraídas de la web oficial de ATRESMEDIA referentes a los programas de El Hormiguero donde se realizó el concurso.

4.2 Conjunto de datos

El primer paso realizado para abordar el problema es la confección de un conjunto de datos acorde a la temática planteada. Se propone como objetivo conseguir entre 40 y 100 imágenes de entrenamiento y entre 10 y 30 imágenes de validación para cada uno de los objetos. Los objetos en el problema son los números del 0-9 como se comentó en la descripción del problema.

Para conseguir esta cantidad de imágenes se sigue un tutorial [36] para descargar imágenes desde Google Images. En los próximos párrafos se detallan los pasos a seguir para descargar las imágenes de la clase correspondiente al número cero.

En primer lugar, se ha de acceder a la página de Google Images y realizar una búsqueda como por ejemplo " número cero". Una vez se obtienen los resultados hay que abrir la consola JavaScript del navegador, desde la opción del menú Herramientas y seguidamente Consola Javascript. En la consola hay que introducir las siguientes líneas de código (ver Figura 4.1) para añadir la librería Jquery y así poder acceder a los elementos disponibles en el código HTML de la web de Google Images:

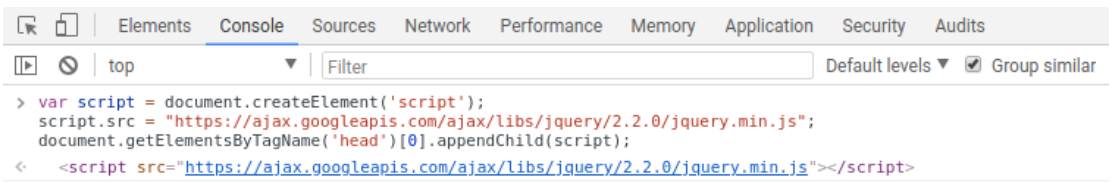

Figura 4.1: Consola JavaScript de Google Chrome.

El siguiente paso es extraer el enlace de la imagen para posteriormente acceder a este y realizar la descarga. Cada imagen está asociada a un componente "div" donde

se encuentra un objeto JSON con información de la imagen como el enlace de esta. En el objeto se encuentra una clave llamada “ou” con el enlace que se está buscando. Para extraer cada uno de estos elementos se debe acceder a cada uno de los componentes “div” de las imágenes y extraer la información del objeto JSON. Los elementos extraídos se almacenan en una lista para posteriormente poder escribirlas en un fichero de texto.

Una vez se han extraído todos los enlaces disponibles para las imágenes se han de almacenar en un fichero de texto para después poder descargar cada imagen desde el enlace correspondiente. Para realizar esta tarea se crea un nuevo componente de tipo enlace “a”, donde se añaden todos los enlaces que se han almacenado en la lista. Por último solo queda simular un clic en el enlace desde la consola y se realiza una descarga con el fichero con todos los enlaces de las imágenes.

Con el archivo de enlaces listos, se implementa un pequeño *script* en Python encargado de descargar cada una de las imágenes. Para inicializar el *script* se pasan como parámetros el fichero que contiene los enlaces y el directorio de salida donde se almacenan las imágenes. Las imágenes se guardan en formato JPG y una vez descargadas se comprueba que cada una de estas sea válida con la librería OpenCV, en caso de no ser válida se elimina del directorio.

En el momento que se obtienen todas las imágenes se ha de crear la estructura correcta para el conjunto de datos. Para ello se hace uso de la herramienta implementada (ver Figura 4.2), donde se eligen un total de 10 clases a crear y los directorios de entrenamiento y validación. Por cada clase se añaden un número de imágenes para validación y entrenamiento, asignando el 80 % y 20 % de los datos respectivamente.

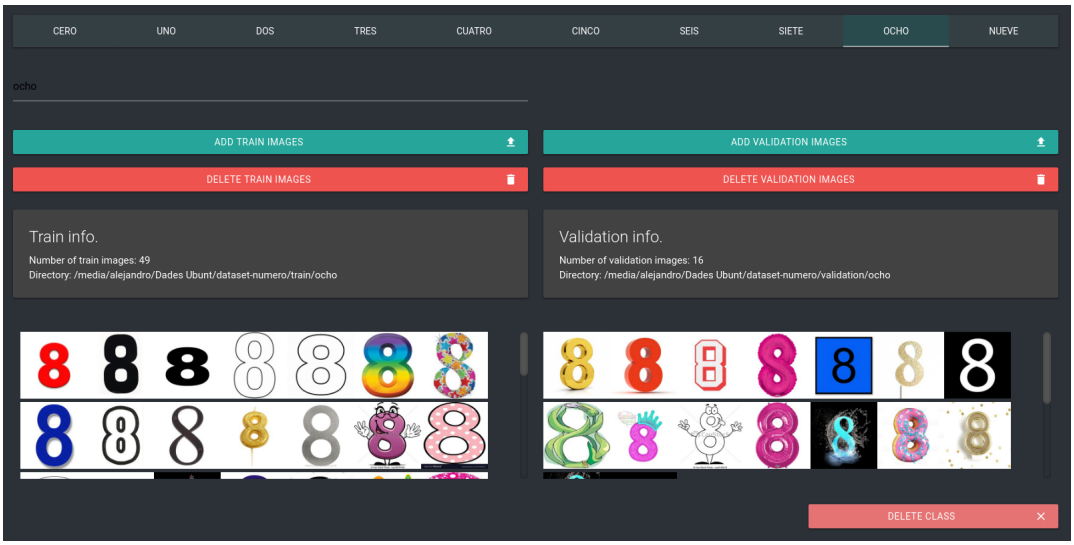

Figura 4.2: Conjunto de datos correspondiente a los números 0-9.

Se obtiene un conjunto de datos con un total de 511 imágenes de entrenamiento y 185 de validación. Las redes convolucionales requieren de conjuntos de datos de miles de imágenes para realizar un buen entrenamiento y obtener buenos resultados, aunque existen opciones cuando el conjunto de datos es pequeño. En la próxima sección se explican las diferentes alternativas empleadas para el entrenamiento del conjunto de datos creado.

Con el procedimiento seguido se comprueba la facilidad de crear un conjunto de datos. En este caso no se ha creado un conjunto amplio debido a que no se han encontrado más imágenes con el buscador Google, pero para realizar unas primeras pruebas puede ser suficiente. Además de Google existen otras posibilidades para crear un conjunto de

datos como el servicio ofrecido por Microsoft para descargar imágenes del buscador Bing [37].

4.3 Entrenamiento del conjunto de datos

Cuando se quiere entrenar una red puede surgir un problema debido a que el conjunto de datos no es lo suficientemente grande para obtener unos resultados óptimos. Por ello se decide hacer uso de un enfoque de la transferencia de aprendizaje explicada en el Capítulo 2.ImageNet[2][22].

El conjunto de datos para el problema planteado se relaciona directamente con el Caso 2 mencionado en la Tabla 2.2. Para este tipo de escenarios es habitual congelar[3] las primeras capas del modelo que se encargan de extraer las características más genéricas como los bordes de una imagen. El resto de capas se vuelven a entrenar y se añade una capa de salida acorde al conjunto de datos que se va a entrenar.

Para la resolución del problema se ha elegido el modelo de red VGG16. En los siguiente apartados se explica el modelo elegido, las configuraciones empleadas para este modelo y los resultados obtenidos.

4.3.1. Modelo empleado para el entrenamiento

El modelo VGG16 fue creado por K. Simonyan y A. Zisserman de la universidad de Oxford. El modelo está entrenado con alrededor de 15 millones de imágenes y clasifica en 1000 clases diferentes [38].

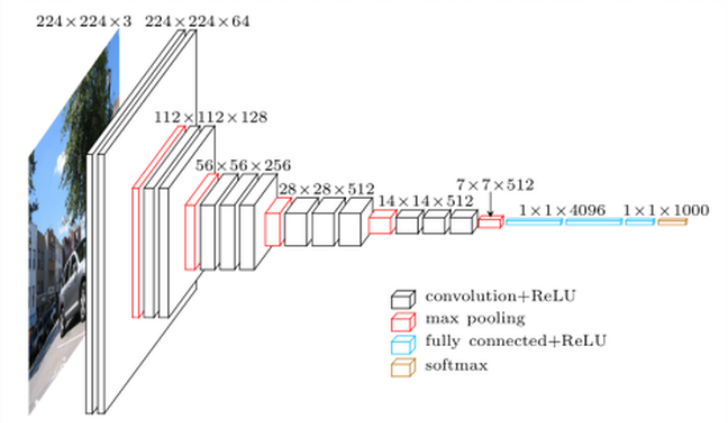

Figura 4.3: Arquitectura VGG16. Imagen extraída de [39].

La entrada de la red está formada por imágenes con unas dimensiones de 224 x 224 píxeles y 3 canales. La primera capa convolucional utiliza un filtro de profundidad igual al de la imagen de entrada. En su arquitectura se componen de 12 capas convolucionales más, donde el filtro de cada una de estas va acorde con la salida de la capa anterior [38].

Además de las capas convolucionales está formada por 5 capas Max Pooling que se encargan de reducir las dimensiones de la imagen hasta una imagen de 7x7x512 como se observa en la Figura 4.3. La salida está conectada a 2 capas totalmente conectadas de 4096 neuronas y un clasificador para 1000 clases [38].

[2]Se trata de un conjunto de datos para el uso en la investigación de la detección de objetos en imágenes.
[3]Se refiere a mantener los pesos cargados de la red preentrenada.

Como se explicó en el capítulo de la herramienta desarrollada, es posible realizar un entrenamiento a partir de una red preentrenada gracias a la API de Keras. Para la configuración de un entrenamiento a partir de una red preentrenada había que aportar una salida acorde al conjunto de datos que se quería entrenar. El fichero creado correspondiente a la salida está compuesto por:

- Una capa *Flatten* que se encarga de "aplanar" el resultado de las capas convolucionales para poder utilizarlo como un vector de características en las capas totalmente conectadas.
- Una capa densa con un total de 1024 neuronas totalmente conectadas con la anterior capa y una función de activación ReLU.
- Una capa densa (salida) con un total de 10 neuronas y una función de activación *softmax* para la clasificación de las clases.

Debido a que la entrada está formada por imágenes de 224 x 224 x 3, previamente se han redimensionado las imágenes del conjunto de datos creado. Para realizar esta tarea se ha creado un *script* utilizando la librería OpenCV que automatiza la redimensión de todas las imágenes del conjunto.

El número de capas congeladas, la función de error y el optimizador y sus parámetros se detallan en el siguiente apartado donde por cada una de las configuraciones empleadas se detallan los resultados obtenidos.

4.3.2. Experimentos y resultados

En este apartado se detallan las configuraciones realizadas para el entrenamiento del conjunto de datos a partir del modelo de red preentrenado y los resultados obtenidos para cada uno de los entrenamientos.

Entrenamiento y resultados (I)

Una vez se ha definido el modelo de red se ha de añadir un optimizador y una función de error para compilar el modelo. Como se ha visto en el apartado de la herramienta, existen diferentes optimizadores y funciones de error. Después de leer acerca de optimizadores se ha encontrado que el optimizador más utilizado en la actualidad por su buen rendimiento es Adam [40]. El optimizador RMSprop y Adadelta ofrecen resultados óptimos, pero otra opción interesante y recomendada por sus resultados es SGD + Nesterov Momentum. En este primer entrenamiento se va a utilizar el optimizador Adam sin entrar en más detalles del resto de optimizadores.

La configuración empleada para el primer entrenamiento es la siguiente:

- **Capas congeladas**: se congelan las 10 primeras capas en las cuales se obtienen las características más genéricas como pueden ser los bordes. El resto de capas del modelo son entrenables.
- **Optimizador**: Adam. Parámetros:
  - *Learning rate*: 0.001
  - *beta_1*: 0.9
  - *beta_2*: 0.999

- *Decay*: 0.0
- *epsilon*: 10e-8

- **Función de error**: *categorical_crossentropy*.

El valor de los parámetros para el optimizador Adam se han establecido a partir de los valores recomendados en el artículo de Adam [40], los cuales fueron los que presentaron mejores resultados. Para el primer entrenamiento se establecen un total de 20 iteraciones con un *batch size* de 20.

Resultados

Para exponer los resultados se presentan dos gráficas con los resultados obtenidos durante las 20 iteraciones y una tabla con los 5 mejores resultados obtenidos. En las gráficas se observan los resultados obtenidos tanto para el conjunto de entrenamiento como el de validación. El entrenamiento de la red tiene una duración de 24 minutos y 35 segundos.

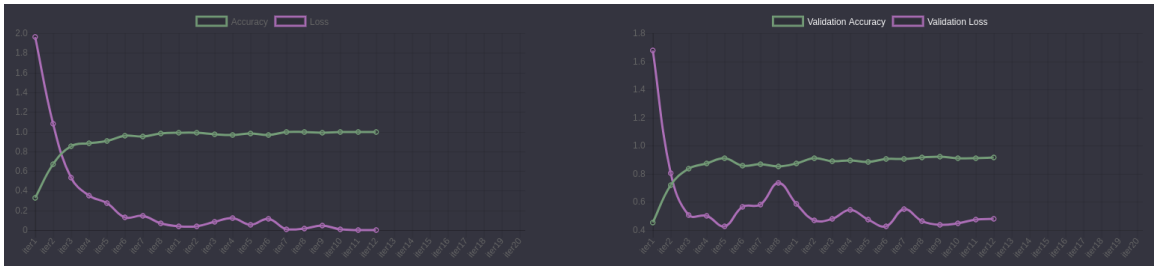

Figura 4.4: Resultados obtenidos para el conjunto de entrenamiento y validación (I).

En la Figura 4.4, a la izquierda, se visualiza la gráfica correspondiente al conjunto de entrenamiento. En esta primera gráfica se observa como la tasa de error ("*loss*", en morado) va descendiendo en cada iteración y como la tasa de acierto ("*accuracy*", en verde) a su vez va aumentando. Por otra parte, en la parte derecha se encuentra la gráfica del conjunto de validación donde el error se encuentra en picos altos en las primeras iteraciones y después va descendiendo. El acierto en el conjunto de validación también va ascendiendo pero muy lentamente. Para ver estos resultados más claramente en la Tabla 4.2 se aportan los 5 mejores resultados obtenidos.

Iteración	Loss	Accuracy	Validation Loss	Validation Accuracy
5	0.2725	0.9075	0.4276	0.9135
10	0.0413	0.9877	0.4698	0.9124
16	0.0139	0.9979	0.4637	0.9145
17	0.0439	0.9858	0.4383	0.9197
20	0.0001	1	0.4783	0.9145

Tabla 4.1: Cinco mejores resultados obtenidos durante el entrenamiento.

Aquí se ha podido comprobar que la función del conjunto de validación es indicar como se esta comportando el modelo en cada fase y en este caso comprobar si el modelo esta sufriendo un sobreajuste. Con esta información es el desarrollador del modelo el que ha de realizar modificaciones en los hiperparámetros del modelo para conseguir unos resultados diferentes.

Con los resultados obtenidos podemos concluir que se ha conseguido un buen *accuracy* para el conjunto de validación pero tambíen se observa que el error para este conjunto va aumentando. El aumento de el error está indicando que hay un sobreajuste o

overfitting. En el siguiente entrenamiento se modifican algunos parámetros para intentar reducir el sobreajuste y reducir el error para el conjunto de validación.

Entrenamiento y resultados (II)

En el primer entrenamiento no se han conseguido malos resultados si hablamos en términos de acierto. Sin embargo la tasa de error para el conjunto de validación ha indicado que el modelo sufre un sobreajuste que hay que intentar mitigar. Para esto se realizan las siguientes modificaciones:

- En el fichero de las capas totalmente conectadas, se añade otra capa densa con un total de 1024 neuronas totalmente conectadas y una función de activación ReLU .
- En el mismo fichero se va hacer uso de la técnica *dropout* que consiste en desactivar aleatoriamente un número prefijado de neuronas en cada una de las capas densas, menos en la capa de salida.
- Se va a establecer un valor más pequeño al valor de la tasa de aprendizaje para el optimizador Adam.

Después de las modificaciones realizadas la configuración final es:

- **Capas congeladas**: se congelan las 10 primeras capas en las cuales se obtienen las características más genéricas como pueden ser los bordes. El resto de capas del modelo son entrenables.
- **Optimizador**: Adam. Parámetros:
  - *Learning rate*: 0.0001
  - *beta_1*: 0.9
  - *beta_2*: 0.999
  - *Decay*: 0.0
  - *epsilon*: 10e-8
- **Función de error**: *categorical_crossentropy*.

En este entrenamiento se amplían el número de iteraciones a 30 y se mantiene el *batch size* a 20.

Resultados

Siguiendo la estructura del primer entrenamiento, se presentan dos gráficas y una tabla de resultados. El tiempo de entrenamiento ha tenido una duración de 40 minutos y 20 segundos.

En la Figura 4.5 se observan los resultados en forma de gráfica. La gráfica situada a la izquierda correspondiente al conjunto de entrenamiento presenta resultados similares al del primer entrenamiento, donde se observa que el error va disminuyendo y la tasa de acierto va aumentando manteniéndose estable. Por otro lado, la gráfica de la derecha también presenta resultados similares, dando a entender que no se han resuelto los problemas de sobreajuste. Aunque se ha aumentado un poco la tasa de acierto, la tasa de error ha incrementado en algunos picos. Este aumento también puede deberse al incremento de iteraciones.

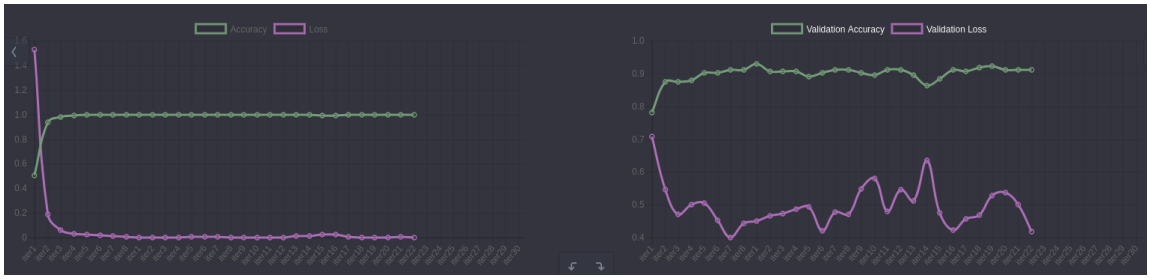

Figura 4.5: Resultados obtenidos para el conjunto de entrenamiento y validación (II).

En la Tabla 4.2 se aportan los 5 mejores resultados obtenidos durante el entrenamiento. En este caso se puede observar como la tasa de error para el conjunto de validación es relativamente alta en comparación a la del conjunto de entrenamiento, indicando de esta manera el sobre ajuste en la red. Se alcanza una tasa de acierto ("*validation accuracy*") del 92 % siendo este un resultado óptimo.

Iteración	Loss	Accuracy	Validation Loss	Validation Accuracy
8	0.0052	0.9989	0.4441	0.9120
9	0.0024	1	0.4494	0.9285
26	0.0011	1	0.4680	0.9175
27	0.0016	0.9994	0.5283	0.9230
28	0.0005	1	0.5375	0.9125

Tabla 4.2: Cinco mejores resultados obtenidos durante el segundo entrenamiento

Después de realizar dos entrenamientos con el optimizador Adam no se han obtenidos malos resultados, aunque el problema del sobreajuste debe intentar solucionarse. Es posible que el optimizador Adam no sea bueno para el conjunto de datos por motivos de tamaño de este o que no se hayan encontrado los valores óptimos para los parámetros del optimizador. Como se menciona al principio del primer entrenamiento existen otras alternativas a Adam como el SGD + Nesterov *Momentum* que puede que ofrezca mejores resultados.

Entrenamiento y resultados (III)

En este nuevo entrenamiento se va a realizar un entrenamiento con el optimizador SGD + Nesterov *Momentum* junto con la función de error *categorical_crossentropy*. Con este se intenta obtener mejores resultados que mitiguen el sobreajuste en el modelo.

A continuación se muestra la configuración completa para el modelo de red:

- **Capas congeladas**: se congelan las 10 primeras capas en las cuales se obtienen las características más genéricas como pueden ser los bordes. El resto de capas del modelo son entrenables.
- **Optimizador**: SGD. Parámetros:
  - *Learning rate*: 0.0001
  - *Decay*: 1e-6
  - *Momentum*: 0.9
  - *Nesterov*: True

- **Función de error**: *categorical_crossentropy*.

El valor de los parámetros utilizados son los valores por defecto establecidos en Keras a excepción de la tasa de aprendizaje que se ha reducido de 0.1 a 0.0001 debido a que en una primera prueba con esta tasa se obtienen unos valores elevados para la tasa de error. En este entrenamiento se amplían el número de iteraciones a 50 y se mantiene el *batch size* a 20.

Resultados

Se sigue la estructura de los dos primeros entrenamientos, y se presentan dos gráficas de resultados y una tabla. El tiempo de entrenamiento ha tenido una duración de 13 minutos y 40 segundos siendo este muy inferior al obtenido en los primeros entrenamientos.

En las nuevas gráfica de la Figura 4.3 se observa un escenario totalmente diferente al de los anteriores entrenamientos. Para el conjunto de entrenamiento se observa como la tasa de acierto y error no convergen rápidamente a valores cercanos a 1 y 0 lo que está indicando que la red no está sufriendo un sobreajuste. Por otro lado para el conjunto de validación la situación también ha cambiado, donde el principal problema que se tenía con los valores de la tasa de error que no lograban descender en este caso si se ha solucionado.

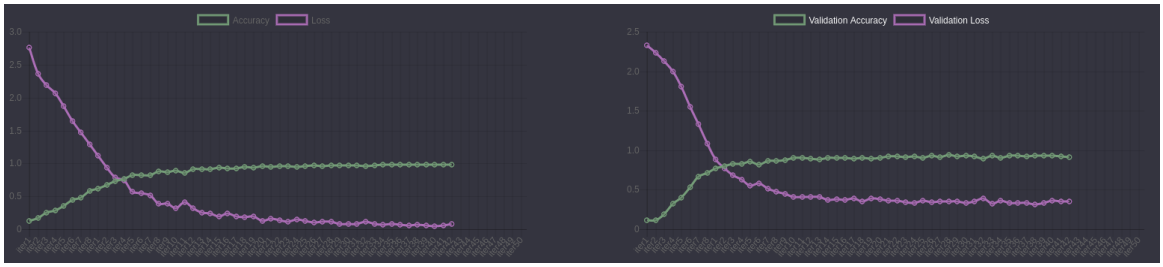

Figura 4.6: Resultados obtenidos para el conjunto de entrenamiento y validación (III).

En la Tabla 4.3 se aportan los 5 mejores resultados obtenidos durante el entrenamiento. En este caso se puede observar como la tasa de error para el conjunto de validación va descendiendo en cada iteración y no tiende a subir, indicando de esta manera que el problema del sobreajuste se ha mitigado. Otros de los motivos por el que no se observa el sobreajuste, es que tanto la tasa de acierto como la de error no han convergido a valores muy cercanos a 1 y 0 respectivamente.

Iteración	Loss	Accuracy	Validation Loss	Validation Accuracy
29	0.1619	0.9432	0.3632	0.9175
34	0.0996	0.9667	0.3446	0.9285
36	0.1132	0.9686	0.3552	0.9480
38	0.0829	0.97455	0.3329	0.9285
46	0.0736	0.9765	0.3184	0.9340

Tabla 4.3: Cinco mejores resultados obtenidos durante el tercer entrenamiento.

Para este entrenamiento podemos observar en la Tabla 4.3 que se ha conseguido una tasa de acierto muy cercana al 93 % y que la tasa de error para el conjunto de validación va decreciendo en cada iteración. Aunque el problema del sobreajuste parece que se ha mitigado en cierta manera, ahora se observa que el aprendizaje es lento y que el error se

estabiliza cerca del 32 %. Aunque estos resultados son óptimos se va a realizar un nuevo entrenamiento aumentando el valor de la tasa de aprendizaje para intentar acelerar el proceso de aprendizaje del modelo y reducir el error de validación.

Entrenamiento y resultados (IV)

En este entrenamiento se va a aumentar ligeramente la tasa de aprendizaje con el fin de reducir el error obtenido en el conjunto de validación.

A continuación se muestra la configuración completa para el modelo de red:

- **Capas congeladas**: se congelan las 10 primeras capas en las cuales se obtienen las características más genéricas como pueden ser los bordes. El resto de capas del modelo son entrenables.
- **Optimizador**: SGD. Parámetros:
  - *Learning rate*: 7e-4
  - *Decay*: 1e-7
  - *Momentum*: 0.9
  - *Nesterov*: True
- **Función de error**: *categorical_crossentropy*.

El número de iteraciones y *batch size* es de 50 y 20 respectivamente, igual que en el entrenamiento anterior.

Resultados

El tiempo de entrenamiento ha tenido una duración de 12 minutos y 35 segundos. En la Figura 4.7 se puede observar un comportamiento similar al del anterior entrenamiento. El objetivo de este nuevo entrenamiento era reducir la tasa de error para el conjunto de entrenamiento y se ha conseguido reduciendo este error en algunos puntos. Por otro lado aunque en la gráfica no se pueda apreciar se ha mejorado la tasa de acierto. Para poder observar con detalle estos resultados se adjuntan en la Tabla 4.4.

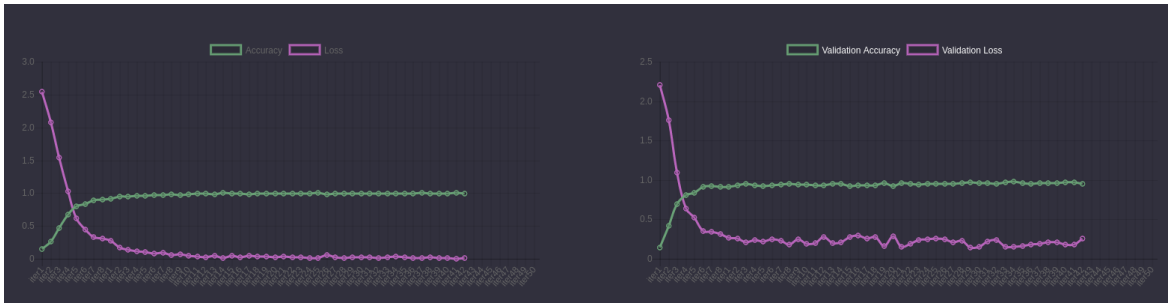

Figura 4.7: Resultados obtenidos para el conjunto de entrenamiento y validación (IV).

En la Tabla 4.4 se incluyen las 5 iteraciones que mejores resultados han producido durante el entrenamiento. Si se observa el conjunto de validación, la tasa de error se ha reducido considerablemente respecto a los resultados de la Tabla 4.3. También los resultados para la tasa de acierto que se ha alcanzado cerca de 97 % superando así a la del anterior entrenamiento.

Iteración	Loss	Accuracy	Validation Loss	Validation Accuracy
38	0.0226	0.9913	0.1535	0.9627
41	0.0188	0.99347	0.1546	0.9680
42	0.0297	0.9891	0.1515	0.9787
48	0.0108	0.9978	0.1803	0.9680
49	0.0041	1	0.1836	0.9734

Tabla 4.4: Cinco mejores resultados obtenidos durante el cuarto entrenamiento.

En este entrenamiento se ha conseguido reducir finalmente la tasa de error para el conjunto de validación. Se ha reducido aproximadamente hasta un 15 % que posiblemente con el incremento de algunas iteraciones más hubiera ofrecido un resultado ligeramente mejor. La consecución de esta reducción ha llevado también a conseguir un tasa de acierto superior a la que se había conseguido hasta el momento.

Después de realizar de realizar 4 entrenamiento con diferentes configuraciones para un modelo de red, en este último entrenamiento se concluye que el modelo se comporta mejor con el optimizador SGD + Nesterov *Momentum* que con Adam para las configuraciones aportadas. Además se ha conseguido una tasa de acierto cercana al 97 % por lo que este resultado podría ser óptimo para solucionar una parte del problema. Para comprobar la efectividad del modelo se realizan una serie de pruebas de *test* cuyos resultados se recogen en la Tabla 4.5.

Imagen de *test*	Predicción (%)
	9.9931e-01
	9.9999e-01
	1.000
	1.000
	1.000
	1.000
	1.000
	9.9999e-01
	9.9993e-01
	9.9999e-01

Tabla 4.5: Predicción para los números del problema.

En la Tabla 4.5 se observa que el modelo de red ofrece unos buenos resultados a la hora de realizar una predicción para cada una de las imágenes de *test*. El valor de la predicción se corresponde con la etiqueta de clase del número correspondiente en cada fila. Con la efectividad conseguida para la resolución de esta parte del problema se da por concluido el apartado de entrenamientos y se da paso a los siguientes subproblemas.

4.4 Detección de objetos

Tal y como se explicó en la descripción del problema se han de detectar nueve objetos diferentes que se corresponden a una línea de teléfono. Estos números siempre mantienen el mismo formato sí se habla de tipografía o color, en cambio el tamaño puede variar en cada concurso realizado. Las Figuras 4.8 y 4.9 son dos ejemplos de imágenes de las cuales se ha de extraer el número de teléfono.

Figura 4.8: Imagen con número de teléfono a extraer (I).

Figura 4.9: Imagen con número de teléfono a extraer (II).

Para la extracción de los números se han utilizado dos técnicas diferentes para la detección de objetos en imágenes en visión por computador. Se implementa la técnica de la ventana deslizante y la de detección de contornos. La razón de la elección de estas técnicas se debe a la forma y la distribución de los objetos en las imágenes. Existen otras técnicas como HaarCascade con OpenCV, redes convolucionales basadas en regiones o el algoritmo YOLO (*You Only Look Once*). En el Apéndice C se relata cómo realizar un entrenamiento HaarCascade con OpenCV, aunque para este problema no se haya empleado finalmente.

4.4.1. Ventana deslizante

El algoritmo de la ventana deslizante es un algoritmo de fuerza bruta debido que no se sabe si en la imagen que se está analizando se encuentran o no los objetos que se buscan. Se implementa una ventana con un tamaño ajustable que se desplaza por toda la imagen en busca de objetos [41]. La complejidad del algoritmo está basada en el tamaño de la imagen siendo ésta mayor cuanto más grande es la imagen.

Para este tipo de técnicas se han desarrollado dos algoritmos diferentes, uno que realiza la búsqueda horizontal en la imagen y otro que realiza la búsqueda en vertical y horizontal.

Ventana deslizante horizontal

Si se observan detenidamente las imágenes 4.8 y 4.9 se comprueba que los objetos que se quieren detectar se encuentran a partir de mitad (altura) de la imagen. Si se obvia la mitad de la imagen se reduce la complejidad del problema a la mitad, suponiendo

esto una gran mejora para el algoritmo. Además del tiempo de recorrer toda la imagen se suma el tiempo de realizar la predicción para detectar si se trata de un objeto. La predicción se realiza con el modelo de red entrenado y se aplica a cada imagen extraída de la ventana deslizante.

En la Tabla 4.6 se realiza una comparativa del tiempo de ejecución entre una imagen completa y la mitad de esta. La imagen tiene unas dimensiones de 400x400 píxeles y se implementa una una ventana de dimensiones de 20 x 80 píxeles. El tamaño de la ventana se establece en relación a las dimensiones de los objetos en las imágenes.

	Tiempo de ejecución
Imagen completa	231.63 segundos
Mitad de la imagen	14.23 segundos

Tabla 4.6: Comparación de resultados para la detección de objetos en una imagen y la mitad de esta.

El algoritmo para los resultados obtenidos en la Tabla 4.6 finaliza cuando se encuentran los nueve objetos correspondiente a un número de teléfono. El primer paso que ha de seguir el algoritmo es encontrar el número "6" en la imagen, si este es encontrado se marca una bandera y se buscan el resto de números en la misma dirección. Si no se encuentran el resto de números se sigue la búsqueda en la siguiente línea y el número '6" se marca como no encontrado.

```
If(!encontrado_seis and prob_numero > 0.9 and num_imagen==6)
    !encontrado_seis=true
    numero[posicion]=6
    posicion+=1
ElseIf(encontrado_seis and prob_numero > 0.9)
    numero[posicion]=num_imagen
    posicion+=1
EndIf
```

Este simple algoritmo para encontrar los números en la imagen también se utilizará para los próximos apartados de ventana deslizante y detección de contornos.

Como conclusión de esta primera prueba, se descarta la búsqueda de los objetos en toda la imagen debido al consumo de tiempo en comparación a la búsqueda a partir de la mitad de la imagen. Sin embargo este segundo algoritmo aún tiene margen de mejora, ya que si se observan las imágenes 4.8 y 4.9 el primer objeto correspondiente al número "6" siempre aparece antes de la mitad (ancho) de la imagen. Se replantea el algoritmo estableciendo que, si la ventana deslizante llega a la mitad de la imagen (ancho) y no encuentra el objeto buscado entonces se descarta la línea y se vuelve a empezar la búsqueda en la siguiente línea. En la Tabla 4.7 se muestran los resultados obtenidos.

	Tiempo de ejecución
Ancho completo	14.23 segundos
Mitad de ancho	7.22 segundos

Tabla 4.7: Resultados del algoritmo de búsqueda con ventana deslizante horizontal.

La nueva implementación reduce el tiempo de búsqueda a la mitad obteniendo 7'22 segundos. Se ha mejorado notablemente el tiempo para encontrar el número completo

pero este todavía sigue siendo un mal resultado para la resolución del problema. El tiempo medio que tarda una persona en visualizar un número y marcarlo está alrededor de los 4 segundos. Para intentar reducir el tiempo obtenido se implementa un nuevo algoritmo que combina una ventana deslizante vertical y horizontal.

Como inconveniente a este algoritmo se encuentra el tamaño de la ventana deslizante. En los diferentes concursos han presentado los objetos con unas dimensiones diferentes. Para centrar un objeto en una ventana esta tiene que tener unas dimensiones acordes al del objeto de la imagen. Otro de los problemas aparecidos es cuando se encuentra un objeto, si el desplazamiento de la ventana es mínimo este vuelve a realizar una predicción para este objeto. Como solución a esto último mencionado, si se encuentra un objeto se desplaza la venta en horizontal un número de píxeles acordes al ancho de los objetos. Se tiene en cuenta que el ancho del objeto número "1" es inferior al del resto de objetos.

El código completo utilizado para este algoritmo se puede ver en el Apéndice E.

Ventana deslizante vertical y horizontal

Siguiendo la filosofía de las ventanas deslizantes, en este apartado se implementa un algoritmo que combina la búsqueda en vertical y horizontal. Partiendo de la mejora que se implementó realizando la búsqueda a partir de la mitad (alto) de la imagen se desarrolla el nuevo algoritmo. La nueva idea está basada en buscar el primer objeto correspondiente al número "6" pero de forma vertical. La distancia vertical es justo la mitad a la horizontal y si la ventana llega a la mitad de la altura y no encuentra el objeto deseado se descarta la línea y se sigue con la siguiente línea vertical. Cuando se encuentra el primer numero "6" la búsqueda con la ventana deslizante cambia a dirección horizontal ya que esto indica que el resto de objetos se encuentra en la misma línea. En la Figura 4.10 se observa la forma de actuar del algoritmo.

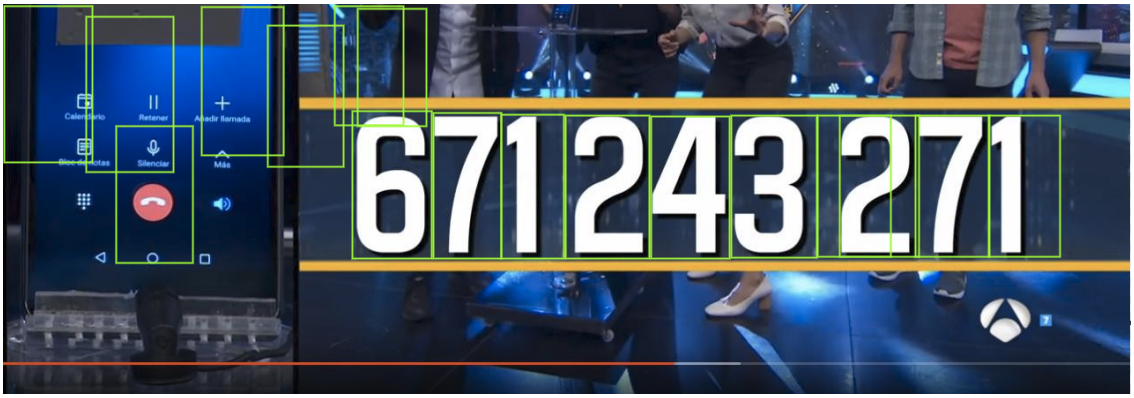

Figura 4.10: Funcionamiento del algoritmo de ventana deslizante vertical y horizontal.

En la Tabla 4.8 se adjunta la comparativa de tiempos entre el algoritmo horizontal que ofrecía mejores resultados y el algoritmo que combina la búsqueda en vertical y horizontal.

	Tiempo de ejecución
Algoritmo horizontal	7.22 segundos
Algoritmo vertical y horizontal	1.74 segundos

Tabla 4.8: Resultados del algoritmo de búsqueda con ventana deslizante vertical y horizontal.

El tiempo conseguido para este nuevo algoritmo ya puede competir contra la habilidad de las personas a la hora de marcar un número de teléfono. Con este nuevo resultado se descarta la implementación de la ventana deslizante horizontal para este problema. Sin embargo, los problemas respecto al tamaño de la ventana deslizante siguen apareciendo. Ajustar la ventana dependiendo el tamaño de la imagen no es una buena solución para el problema planteado debido a que no se puede realizar un ajuste exacto cuando aparezca la imagen en directo, se perdería el tiempo ganado en la búsqueda.

Como conclusión para los algoritmos de ventanas deslizantes, se ha observado que es posible rebajar el tiempo considerablemente desde 230 a 1'74 segundos. Este algoritmo está adaptado al problema que se plantea, por lo que no sería válido para otros problemas. A pesar de los buenos resultados obtenidos, el algoritmo no se considera óptimo por el tema de ajuste de ventanas dependiendo el tamaño de los objetos. Para otros problemas que no se requiera minimizar el tiempo el algoritmo puede ser válido.

El código completo utilizado para este algoritmo se puede ver en el Apéndice E.

4.4.2. Detección de contornos

Se propone un nuevo algoritmo basado en la detección de contornos. La librería OpenCV ofrece diversas funciones para la búsqueda de contornos en una imagen así como funciones para realizar un preproceso a esta. Los pasos a seguir para la detección de contornos correctamente son:

- Pasar imagen a escala de grises.
- Eliminar ruido de la imagen.
- Binarizado de la imagen.
- Búsqueda de contornos.

Estos son los pasos previos que se van a seguir para detectar los contornos, después se calcularán los tiempos de ejecución y las predicciones de los objetos encontrados.

Imagen a escala de grises

Para la conversión a escala de grises es necesario cargar primero la imagen que se quiere procesar. La conversión se realiza para reducir el coste computacional de trabajar con imágenes a color, además para trabajar con la función de detección de contornos es un requisito.

En siguiente código es el encargado de realizar la transformación de la imagen. Se declara una variable llamada "gris" que será la imagen resultante en escala de grises. Para esto se llama a la función "cvtColor" de OpenCV donde como primer parámetro se pasa la imagen original y como segundo el espacio de color al que se quiere transforma la imagen.

```
1 import cv2
2
3 gris = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
4 cv2.imshow(gris)
```

En la figura 4.12 se observa el resultado que se obtiene:

Figura 4.11: Imagen original.

Figura 4.12: Imagen en escala de grises (I).

Filtro Gaussiano

Las imágenes suelen contener un ruido producido entre otros por la iluminación. Existen diferentes técnicas para mitigar el ruido como por ejemplo el filtro Gaussiano. Los filtros aplican convoluciones a la imagen estableciendo un *Kernel* de tamaño N x N que recorre toda la imagen con el objetivo de suavizar la imagen y eliminar el ruido.

En el filtro Gaussiano se observa el valor máximo en el píxel central del *Kernel* disminuyendo hacia los extremos a partir del parámetro de desviación típica. OpenCV tiene implementada una función que calcula este tipo de filtro [42]. Para este filtro, el *Kernel* ha de ser un número impar para que el centro sea un único píxel.

```
1 import cv2
2
3 filtroGaussiano = cv2.GaussianBlur(gris, (5, 5), 0)
4 cv2.imshow(filtroGaussiano)
```

En el código la variable "filtroGaussiano" es la imagen resultante de aplicar el filtro a la imagen en escala de grises. A esta última imagen se aplica la función "GaussianBlur" que recibe como parámetros la imagen original a la que se aplicará el filtro, "(5,5)" que se corresponde con el tamaño del filtro y por último la desviación típica. Si en la desviación típica se pasa como parámetro un "0" OpenCV se encarga de aplicar el valor más adecuado.El resultado que se obtendrá es el que se observa en la Figura 4.14.

Figura 4.13: Imagen en escala de grises (II).

Figura 4.14: Imagen con filtro Gaussiano (I).

Binarización de la imagen

La binarización se trata de una segmentación de la imagen. La finalidad es separar los objetos que se plantean en el problema de la imagen del resto de objetos de la imagen. Para realizar la separación de los objetos se compara la intensidad de cada píxel de la imagen con un umbral establecido.

En la binarización de una imagen se aplica un umbral de la siguiente manera [43]:

$$x = \begin{cases} maxVal & \text{si } pixel(x,y) > umbral \\ 0 & \text{otro caso} \end{cases}$$

Si el píxel examinado tiene una intensidad mayor que el umbral establecido a este píxel se establece el valor más grande fijado. En caso contrario el píxel toma valor 0 [43]. El código Python para realizar esta operación es:

```
import cv2

bin = cv2.threshold(filtroGaussiano, 240, 255, cv2.THRESH_BINARY)[1]
cv2.imshow(bin)
```

El código implementa la función "thresholds" de OpenCV donde como primer parámetro recibe la imagen a la que se desea aplicar una segmentación. El segundo parámetro se corresponde con el umbral, que se establece un valor alto debido a que los objetos del problema tendrán un umbral cercano a 255 ya que son de color blanco. El tercer parámetro es el valor máximo que establece a un píxel si este supera el umbral. El último parámetro se indica el tipo de segmentación que se desea utilizar, en este caso, binarización. Los resultados se observan en la Figura 4.17.

Figura 4.15: Imagen con filtro Gaussiano (II).

Figura 4.16: Imagen binarizada (I).

Detección de contornos

Con el método de OpenCV para la detección de contornos se pretende encontrar los objetos planteados en el problema. Un contorno es una línea de puntos donde el principio y el final es el mismo punto, no habiendo puntos vacíos en esta línea.

```
import cv2

contornos = cv2.findContours(bin.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
cv2.imshow(contornos)
```

En el código se aplica la función "findContours" de OpenCV donde:

- En el primer parámetro se pasa la imagen binarizada que se proceso anteriormente.
- En el segundo parámetro se indica el contorno buscado. El valor "RETR_EXTERNAL" indica que se quieren los contornos externos de los objetos.

- En el último parámetros se indican los puntos que se quieren almacenar de un contorno. En este caso se indica que se eliminen los puntos redundantes. Una línea puede estar formada por una gran cantidad de puntos y con almacenar los puntos situados a los extremos sería suficiente.

Figura 4.17: Imágenes binarizada (II).

Figura 4.18: Imagen con detección contornos.

Para dibujar los contornos se recorren todos los contornos almacenados y se van dibujando en la imagen original con la función "drawContour".

Extracción de objetos

Detectados los contornos de los objetos se ha de predecir el resultado para estos, extrayendo cada uno de estos individualmente. Para la extracción de un objeto se tiene en cuenta algunos de los puntos de los contornos de cada objeto detectado. Los pasos a seguir para la extracción a partir de los puntos de contornos son:

- Extraer el punto más alto (*max(y)*).
- Extraer el punto más bajo (*min(y)*).
- Extraer el punto más a la izquierda (*min(x)*).
- Extraer el punto más a la derecha (*max(x)*).

Los puntos extraídos son suficientes para recortar el objeto de la imagen y tratarlo de forma individual. La imagen resultante se redimensiona a 224x224 píxeles que es el tamaño establecido para la entrada del modelo de red entrenado.

Resultados

Detallado el preproceso y extracción que se aplica a las imágenes, se realizan múltiples pruebas para las diferentes imágenes. Como en el apartado de ventanas deslizantes, se realizan pruebas con la imagen completa y la mitad de esta (alto) para comprobar cómo se comporta el sistema con el nuevo algoritmo. En la Tabla 4.9 se recogen los resultados obtenidos.

	Tiempo de ejecución (imagen completa)	**Tiempo de ejecución** (mitad de la imagen)
Imagen 1 (Figura 4.8)	0.2559 segundos	0.2587 segundos
Imagen 2 (Figura 4.9)	0.2521 segundos	0.2508 segundos

Tabla 4.9: Resultados del algoritmo para la detección de contornos

En este nuevo algoritmo el tiempo es el mismo para la imagen completa y la mitad de esta. Esto se debe a que en la parte superior de la imagen no se detecta ningún contorno (ver Figura 4.17) y por lo tanto no se realiza ninguna predicción. En cuestión de tiempos, se obtienen unos resultados excelentes en comparación al mejor resultado obtenido para el algoritmo de la ventana deslizante. También se eliminan los problemas de la ventana deslizante donde se había de establecer una ventana acorde para cada imagen dependiendo las dimensiones de los objetos a buscar.

Problema detectado y propuesta de solución

Aunque los tiempos que se obtienen son excelentes, se detecta un problema con una nueva imagen generada para realizar un test. Esta nueva imagen (ver Figura 4.19) ha sido generada a partir de números existentes en otras imágenes extraídas del programa de televisión.

Figura 4.19: Imagen con objetos a diferentes alturas.

Como se puede observar los objetos están situados a diferentes alturas. El algoritmo anterior para la detección de contornos almacena los contornos de mayor a menor altura. En este caso no se obtiene solución con el algoritmo anterior debido a que cuando detecte el objeto correspondiente al número "6" ya se han detectado otros objetos que no han sido almacenados ya que no eran el número "6".

Para solucionar el problema se añade la librería *imutils*[4], la cual contiene diferentes funciones para trabajar con OpenCV en Python, entre otras la ordenación de contornos. Con esta función los objetos se ordenan de izquierda a derecha [44], eliminando de esta forma el problema encontrado.

Como conclusión, el nuevo algoritmo implementado supera en optimización al resto de algoritmos propuestos. Con el tiempo conseguido es prácticamente imposible que una persona marque nueve números igualando el tiempo conseguido. Con la última mejora implementada se evita el problema para próximos concursos donde los números pueden no estar alineados.

[4]Se trata de una librería con diferentes funciones implementadas para realizar operaciones básicas de OpenCV con Python.

4.5 Detección y predicción de objetos en tiempo real

El preproceso y técnica de detección de contornos descritos se realiza sobre una única imagen. Para el problema, se plantea la detección de los objetos cuando estos aparezcan en la pantalla de televisión, es decir, en tiempo real. OpenCV ofrece un método para la captura de vídeo, desde el cual se puede acceder a archivos de vídeo almacenados en la máquina o haciendo uso de una *webcam*.

Se han realizado pruebas con ambas funciones para la lectura de vídeos, donde se ha comprobado que la lectura desde ficheros almacenados en la máquina produce mejores resultados. Esto se debe a que la *webcam* utilizada no captura las imágenes con la misma calidad que las imágenes extraídas desde un fichero de vídeo. Sin embargo para el problema se utiliza la opción de la *webcam* para capturar la imagen de la televisión, obteniendo resultados satisfactorios.

La máquina desde la que se hace uso del *script* programado para la detección de objetos, es capaz de capturar un total de 30 FPS (*frames* por segundo). Para cada uno de estos *frames* se pasa el algoritmo de detección de objetos y se realizan las posibles predicciones para cada objeto detectado. Este proceso ralentiza la reproducción del vídeo ya que en un segundo puede realizar cerca de 100 predicciones. Esto se debe a que antes de que aparezcan los objetos correspondientes a los números de teléfono, pueden aparecer otra serie de objetos diferentes pero con características de color similares que hace que su contorno sea detectado también. Para mitigar este problema se reduce el número de *frames* a tratar de 30 a 3 aproximadamente para el preproceso y reconocimiento de objetos. Con esta modificación se pueden perder algunas décimas de segundo.

En el apartado de ventana deslizante se aporta un pequeño pseudo-código referente al algoritmo para encontrar los números en una imagen donde para cada *frame* procesado se hace uso de este. A continuación se adjunta el código utilizado en Python para la detección en tiempo real de un número de teléfono donde se incluye también el pseudo-código visto con anterioridad en el apartado de ventanas deslizantes:

```
def procesar_contornos(image,clases):
  global numero
  #flag para comprobar si se ha encontrado el primer 6
  primero=0
  #posicion del objeto encontrado
  pos=0
  #pasar imagen a escala de grises
  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  blurred = cv2.GaussianBlur(gray, (5, 5), 0)
  #aplicar filtro gaussiano
  thresh = cv2.threshold(blurred, 240, 255, cv2.THRESH_BINARY)[1]
  #buscar contornos en la imagen binarizada
  contorno = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
   cv2.CHAIN_APPROX_SIMPLE)
  contornos = contornos[0] if imutils.is_cv2() else contornos[1]
  #ordenacion de contornos de izquierda a derecha
  if(len(contornos)>0):
    (contornos, _) = contours.sort_contours(contornos)
  #recorrer lista de contornos
  for c in contornos:
    #punto mas a la izquierda
    extLeft = tuple(c[c[:, :, 0].argmin()][0])
    #punto mas a la derecha
    extRight = tuple(c[c[:, :, 0].argmax()][0])
    #punto mas alto
    extTop = tuple(c[c[:, :, 1].argmin()][0])
```

```
27      #punto mas bajo
28      extBot = tuple(c[c[:, :, 1].argmax()][0])
29      #recortar imagen a partir de los puntos
30      recorte = image[extTop[1]-10:extBot[1]+10, extLeft[0]-10:extRight[0]+10]
31      h,w,can=recorte.shape
32      #se comprueba que la figura no sea una linea y tenga similitud a un n mero
33      if(w >25 and h>45 ):
34        #redimension de la imagen a las dimensiones de la red entrenada
35        recorte = cv2.resize(recorte, (224, 224))
36        recorte = recorte.astype("float") / 255.0
37        recorte = img_to_array(recorte)
38        recorte = np.expand_dims(recorte, axis=0)
39        #prediccion de la imagene
40        prediccion = model.predict(recorte)
41
42        if(not(primero) and max(prediccion[0])>0.99 and clases[prediccion[0].
              tolist().index(max(prediccion[0]))]==6):
43          numero[0]=clases[prediccion[0].tolist().index(max(prediccion[0]))]
44          primero=1
45          pos+=1
46        elif(primero and max(prediccion[0])>0.99):
47          numero[pos]=clases[prediccion[0].tolist().index(max(prediccion[0]))]
48          pos+=1
49
50        if(numero[8]!='a'):
51          #NUMERO ENCONTRADO
52          telefono = ''.join(str(x) for x in numero[:9])
53          llamada(telefono)
54          break
55
56  def empieza(clases):
57      #cap = cv2.VideoCapture(0)
58      cap= cv2.VideoCapture("video_formiguer_3.mp4")
59
60      # Comprpbar disponibilidad de la camara
61      if (cap.isOpened()== False):
62        print("Error abriendo video")
63
64      fps=0
65      # Leer mientras haya video
66      while(cap.isOpened() and numero[8]=='a'):
67        ret, frame = cap.read()
68        if ret == True:
69          fps+=1
70          if(fps==10):
71              procesar_contornos(cv2.resize(frame, (800, 600)),clases)
72              fps=0
73          cv2.imshow('Frame',cv2.resize(frame, (800, 600)))
74
75  #etiquetas de las clases
76  clases=[0,5,4,2,9,8,6,7,3,1]
77  #lista para numero de telefono
78  numero=['a']*9
```

El código está formado por dos funciones, *empiza* y *procesar_contornos*. La función *empieza* se encarga de reproducir un fichero de vídeo de la máquina o a capturar video a través de la *webcam* y enviar *frames* a la función de *procesar_contornos*. Esta última función es la encargada de aplicar el preproceso a las imágenes. En la línea 18 se recorren todos los contornos localizados en la imagen y en la línea 36 se realiza la predicción para cada objeto resultante de los puntos de contorno . En las líneas 40 y 44 se comprueba que este objeto sea un número. Se establece que lo trate como un número si la probabilidad de este es superior a 0'99 (99 %). Si se establece una probabilidad superior a 0.9 el código

funcionará correctamente, en cambio si está por debajo de esta puede que los resultados no sean exactos y devuelva una predicción incorrecta.

En conclusión, este método para la detección de objetos en tiempo real no es el más correcto ya que como se ha comentado puede ralentizar la predicción. Sin embargo para el propósito de la aplicación ha funcionado correctamente perdiendo alguna décima de segundo. Para otro tipo de problemas sería necesario el uso de técnicas como Haar-Cascade o el uso de técnicas más novedosas para la detección de objetos como las redes convolucionales basadas en regiones y el algoritmo YOLO entre otros.

4.6 Llamada telefónica

El último subproblema a resolver es realizar una llamada al número de teléfono extraído de la imagen. Para ello se implementa una aplicación para dispositivos Android. Se trata de una aplicación básica sin ningún diseño estético y con la única funcionalidad de realizar una llamada. Para la implementación se utiliza el entorno de desarrollo Android Studio[5].

Cuando se obtiene el número de teléfono este ha de llegar hasta el teléfono con el que se va a realizar la llamada. Para que el número sea enviado se ha de establecer una comunicación entre la máquina donde se detecta el número y el teléfono que realiza la llamada. Se ha elegido una comunicación basada en *Sockets* donde el teléfono móvil actúa como servidor y la máquina como cliente. En la Figura 4.20 se muestra el esquema desde que se captura la imagen hasta que se realiza la comunicación:

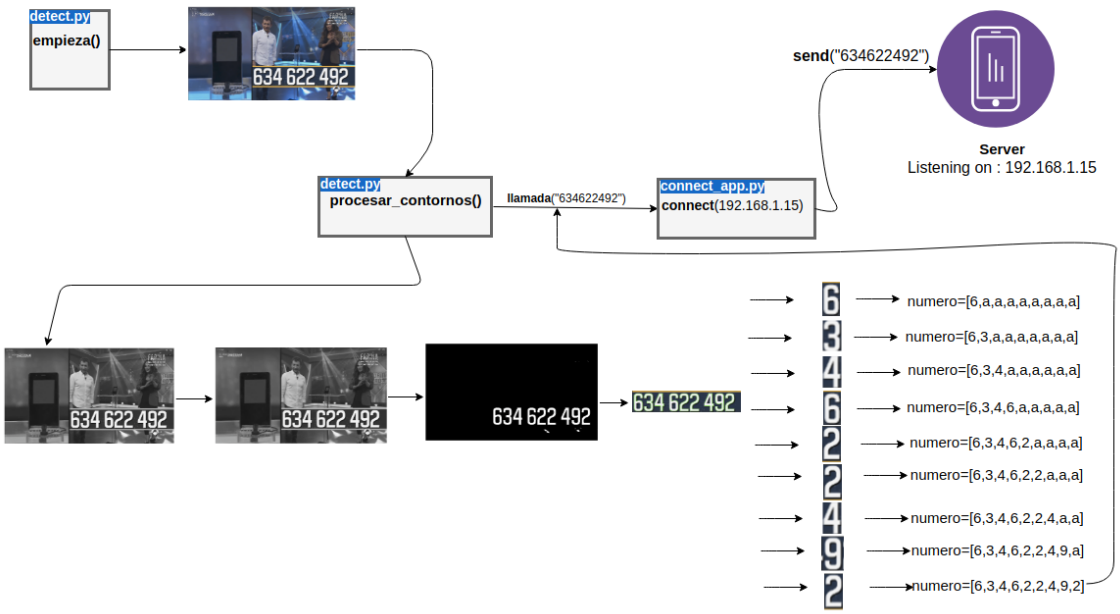

Figura 4.20: Esquema general del problema planteado.

Al abrir la aplicación en el teléfono móvil donde se encuentre instalada, este actúa como un servidor y se pone a la escucha de peticiones en la *IP* que tiene asignada el teléfono en la red a la que esté conectado. El servidor está únicamente preparado para recibir una cadena formada por números. Está cadena se procesa y se pasa a la función específica de Android para realizar llamadas. Hay que mencionar que una aplicación con

[5]Es un entorno de desarrollo integrado oficial para la plataforma Android.

este tipo de funcionalidad no sería válida para la tienda de Google Play Store, debido a que se realiza una llamada directamente sin que el usuario tenga que realizar ninguna acción.

Se implementa una aplicación simple pero que cumple con los requisitos del problema. Se observa como los resultados obtenidos en una máquina específica llegan hasta un destino fijado realizando una serie de comunicaciones. El tiempo total para detectar el número de teléfono y realizar la llamada oscila entre 0'7 y 1'2 segundos, por lo que se concluye que se ha resuelto el problema consiguiendo unos tiempos que mejoran los tiempos que pueda conseguir una persona sin la ayuda de este tipo de tecnologías.

CAPÍTULO 5
Conclusiones

Alcanzar los objetivos propuestos para la realización de este trabajo ha sido una gran satisfacción a nivel personal. Ha sido una tarea compleja debido al escaso tiempo disponible para la realización de un proyecto de este alcance.

En primer lugar se ha conseguido el objetivo de realizar una herramienta con diferentes funcionalidades para realizar modelos basados en el aprendizaje profundo. La herramienta podría contener una gran variedad de opciones más, pero hasta el momento la funcionalidad cumple con los objetivos y requisitos planteados.

Se han encontrado nuevas tecnologías que permiten realizar aplicaciones multiplataforma. Ha sido una buena elección trabajar con la tecnología Atom donde se han integrado diferentes tecnologías a partir de una serie de comunicaciones entre estas. El diseño estético de una herramienta de escritorio siempre es un quebradero de cabeza que en el actual trabajo gracias a la elección mencionada, ha facilitado mucho este proceso.

En segundo lugar, se ha conseguido dar soluciones para un problema planteado. En un primer momento se plantearon diferentes problemas a resolver como la detección de enfermedades en las hojas de las plantas o algún problema relacionado con el reconocimiento facial. Sin embargo surgió un nuevo problema planteado que, después de examinarlo, se concluyó que era válido para realizar un ejemplo que cumpliese con los objetivos del trabajo.

Con el problema planteado sobre la detección de números, se ha demostrado que la herramienta desarrollada es completamente válida para resolver una parte del problema. Con el modelo generado con la herramienta se han podido resolver el resto de problemas a partir de diferentes técnicas para la visión por computador.

Respecto a los resultados obtenidos en la resolución del problema, ha estado interesante probar diferentes técnicas de la visión por computador y ver cómo los resultados iban mejorando en cada nueva propuesta. Empezar con unos resultados de un tiempo cercano a los 4 minutos y reducirlo hasta 1 segundo ha sido una tarea difícil pero que ha ayudado a entender las diferentes técnicas y el porqué se utilizan en la visión por computador.

Por último, es una gran satisfacción aplicar los nuevos conceptos aprendidos sobre redes neuronales convolucionales junto las diferentes técnicas de la visión por computador para resolver un problema que se planteó de forma casual.

5.1 Trabajos futuros

El tiempo ha sido uno de los factores que más en cuenta se ha tenido para el desarrollo de la herramienta. Los trabajos futuros se separan en dos puntos para diferenciar la parte del desarrollo de la herramienta y el problema planteado.

Herramienta

Debido al horario de trabajo en las prácticas de formación en empresa, el tiempo diario se reducía notablemente para realizar un desarrollo más amplio. Sin embargo, se tienen muy en cuenta las ampliaciones que se pueden aplicar a la herramienta.

- **Integrar servicios cloud**. Actualmente diferentes plataformas ofrecen diferentes máquinas virtuales con todas las tecnologías necesarias para el aprendizaje profundo. Ofrecer la posibilidad al usuario de conectar la herramienta a su máquina virtual y realizar los entrenamientos en está. Esto dependiendo de la máquina personal del usuario y la máquina virtual que tenga contratada podría reducir considerablemente los tiempos de entrenamiento.
- **Obtención de imágenes para un conjunto de datos**. En el módulo de creación de un conjunto de datos, se quiere añadir una opción para descargar imágenes a partir de las API's ofrecidas por Google o Microsoft. Esta nueva funcionalidad facilita notablemente el trabajo de búsqueda de imágenes para el conjunto de datos.
- **Facilidad en la creación de redes convolucionales**. En el apartado de la creación de una red convolucional tanto para redes preentrenadas o propias del usuario, sería conveniente añadir una funcionalidad extra a la de añadir un fichero python con un módulo donde el usuario pueda crear una red de forma visual.
- **Posibilidad de cargar una red convolucional**. Añadir una funcionalidad en el módulo de creación de una red convolucional para que el usuario pueda cargar un modelo de red creado previamente con el fin de realizar modificaciones en este.
- **Visualización gráfica de los filtros de las capas intermedias**. En el módulo de monitorización de resultados, sería interesante añadir una función para mostrar de forma gráfica cómo se comportan los filtros de las capas de la red en cada iteración del entrenamiento.

Estos son los trabajos que se han planteado para una futura implementación. Sin embargo, con un estudio más profundo de las necesidades de un usuario avanzado este número de propuestas se ampliarían. Además hay que tener en cuenta que las tecnologías como Keras están siempre recibiendo actualizaciones lo cual es muy posible que aparecerán nuevas propuestas de implementación.

Problema planteado para el reconocimiento y detección de objetos

Aunque las técnicas empleadas para la resolución del problema han ofrecido buenos resultados, ha quedado pendiente utilizar otras técnicas más novedosas. Como trabajo futuro se propone la resolución del problemas con:

- ***Fast R-CNN***. La detección de objetos mediante redes convolucionales basadas en regiones es una de las propuestas que se ha marcado como objetivo futuro. Se trata de una de las técnicas más modernas para la detección de objetos pero que requiere conocimientos avanzados en el campo de las redes convolucionales.
- **YOLO**. Como se ha mencionado en el trabajo se trata de otro algoritmo capaz de detectar múltiples objetos en una imagen en un tiempo reducido.

Bibliografía

[1] Gray, Chris y Hemken, Terry. (2017). Emerging technologies make their mark on public service. Consultado en https://www.accenture.com/us-en/insight-ps-emerging-technologies [Último acceso: Junio 2018].

[2] Google Trends about Deep Learning. Consultado en https://trends.google.com/trends/explore?date=2008-01-01%202018-07-02&q=deep%20learning [Último acceso: Julio 2018].

[3] Coursera. Deep Learning Specialization. Consultado en https://www.coursera.org/specializations/deep-learning [Último acceso: Abril 2018].

[4] Rosebrock, Adrian. (2014). Blog PyImageSearch. Consultado en https://www.pyimagesearch.com/author/adrian/ [Último acceso: Junio 2018].

[5] Viola, P. y Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *10.1109/CVPR.2001.990517*

[6] Ojala, T., Pietikainen, M. y Maenpaa, T. (2001). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *10.1109/TPAMI.2002.1017623*

[7] Dalal, N. y Triggs, B. (2005). Histograms of oriented gradients for human detection. *10.1109/CVPR.2005.177*

[8] Mallick, Satya. (2016). A Brief History of Image Recognition and Object Detection. Consultado en https://www.learnopencv.com/image-recognition-and-object-detection-part1/ [Último acceso: Marzo 2018].

[9] Universitat Autònoma de Barcelona, Coursera. Cascada de clasificadores. Consultado en https://es.coursera.org/learn/deteccion-objetos/lecture/pRnHu/l5-5-cascada-de-clasificadores [Último acceso: Mayo 2018].

[10] Rosebrock, Adrian. (2015). Local Binary Patterns with Python & OpenCV. Consultado en https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/ [Último acceso: Junio 2018].

[11] Mallick, Satya. (2016). Histogram of Oriented Gradients. Consultado en https://www.learnopencv.com/histogram-of-oriented-gradients/ [Último acceso: Junio 2018].

[12] Brownlee, Jason. (2016). Supervised and Unsupervised Machine Learning Algorithms. Consultado en https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms [Último acceso: Junio 2018].

[13] Gall, Richard. (2018). Different types of Machine Learning. Consultado en https://jaxenter.com/basic-introduction-machine-learning-145140.html [Último acceso: Junio 2018].

[14] Dr. Adrian Rosebrock. *Deep Learning for Computer Vision with Python*. PYIMAGESEARCH, First edition, 2017.

[15] François Chollet. *Deep Learning with Python*. First edition, November 2017.

[16] IBM. (2017). El modelo de redes neuronales. Consultado en https://www.ibm.com/support/knowledgecenter/es/SS3RA7_sub/modeler_mainhelp_client_ddita/components/neuralnet/neuralnet_model.html [Último acceso: Marzo 2018].

[17] Wikipedia. Propagación hacia atrás. Consultado en https://es.wikipedia.org/wiki/Propagacion_hacia_atras [Último acceso: Marzo 2018].

[18] Wikipedia. ImageNet. Consultado en https://en.wikipedia.org/wiki/ImageNet [Último acceso: Marzo 2018].

[19] Mallick, Satya. (2018). Number of Parameters and Tensor Sizes in a Convolutional Neural Network (CNN). Consultado en https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/ [Último acceso: Abril 2018].

[20] MathWorks. Cómo crear y entrenar modelos de aprendizaje profundo. (2018). Consultado en https://es.mathworks.com/discovery/deep-learning.html [Último acceso: Junio 2018].

[21] Keras. Documentation for individual models. Consultado en https://keras.io/applications/ [Último acceso: Junio 2018].

[22] Convolutional Neural Networks for Visual Recognition, Stanford University. (2018). Transfer Learning. Consultado en http://cs231n.github.io/transfer-learning/ [Último acceso: Abril 2018].

[23] PipelineAi. PipelineAi Features. Consultado en https://pipeline.ai/features/ [Último acceso: Junio 2018].

[24] Aetros. (2016).Aetros documentation. Consultado en https://aetros.com/docu [Último acceso: Junio 2018].

[25] Electron. About Electron. Consultado en https://electronjs.org/docs/tutorial/about [Último acceso: Abril 2018].

[26] Nodejs. Documentación Node.js. Consultado en https://nodejs.org/es/docs/ [Último acceso: Abril 2018].

[27] JQuery. Librería JQuery para JavaScript. Consultado en http://api.jquery.com/ [Último acceso: Abril 2018].

[28] Python. Introducción Python. Consultado en http://docs.python.org.ar/tutorial/3/real-index.html [Último acceso: Abril 2018].

[29] ZeroMQ. Sistema de mensajería ZMQ. Consultado en http://zeromq.org/ [Último acceso: Abril 2018].

[30] Felipe Ortega, José. (2015). Comunicaciones disponibles en ZMQ. Consultado en http://www.synergicpartners.com/omq-mensajeria-asincrona-alto-rendimiento [Último acceso: Abril 2018].

[31] Materialize. *Framework* Materialize de Google. Consultado en https://www.materializecss.com/ [Último acceso: Abril 2018].

[32] Tensorflow. Información sobre la biblioteca TensorFlow de Google. Consultado en https://www.tensorflow.org/ [Último acceso: Abril 2018].

[33] Buhigas, Javier. (2018). Funcionalidad de la biblioteca TensorFlow de Google. Consultado en https://puentesdigitales.com/2018/02/14/todo-lo-que-necesitas-saber-sobre-tensorflow [Último acceso: Abril 2018].

[34] Keras. Información sobre la *API* Keras. Consultado en https://keras.io/ [Último acceso: Junio 2018].

[35] OpenCV. Información sobre la librería OpenCV. Consultado en https://opencv.org/ [Último acceso: Abril 2018].

[36] Rosebrock, Adrian. (2018). How to create a deep learning dataset using Google Images. Consultado en https://www.pyimagesearch.com/2017/12/04/how-to-create-a-deep-learning-dataset-using-google-images/ [Último acceso: Mayo 2018].

[37] Rosebrock, Adrian. (2017). How to (quickly) build a deep learning image dataset. Consultado en https://www.pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-image-dataset/ [Último acceso: Mayo 2018].

[38] Simonyan, K. y Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556.*

[39] Blierd, Leonard. (2016).Deep CNN and Weak Supervision Learning for visual recognition. Consultado en https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5 [Último acceso: Mayo 2018].

[40] Diederik P. y Kingma, Jimmy Ba. (2015). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980*

[41] Rosebrock, Adrian. (2015). Sliding Windows for Object Detection with Python and OpenCV. Consultado en https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/ [Último acceso: Mayo 2018].

[42] OpenCV. Image Blurring (Image Smoothing). Consultado en https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html [Último acceso: Mayo 2018].

[43] OpenCV. Threshold Binary. Consultado en https://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html [Último acceso: Mayo 2018].

[44] Rosebrock, Adrian. (2018). Sorting Contours. Consultado en https://github.com/jrosebr1/imutils [Último acceso: Junio 2018].

APÉNDICE A
Especificaciones de la máquina utilizada

La herramienta como el problema planteado se han realizado con un ordenador personal. Se trata de un ordenador portátil MSI GL62M donde las principales características son:

- Procesador Intel® Core i7-7700HQ(2.8 GHz, 4 núcleos, 6MB)
- Memoria 8GB DDR4-2400 (1 módulo, ampliable hasta 32GB)
- Disco duro 1TB (SATA) + 256 GB SSD
- Pantalla 15.6"FHD (1920x1080), IPS 45
- Controlador gráfico GeForce® GTX 1050, 4GB GDDR5
- Cámara web, resolución HD (1024x780)

De este ordenador portátil se ha aprovechado principalmente la tarjeta gráfica GeForce, la cual es compatible para configurar TensorFlow GPU. Esta configuración ha permitido reducir los tiempos de entrenamiento en comparación a los entrenamientos realizados con CPU's.

También se ha utilizado la cámara web integrada, la cual ofrece una buena resolución para los vídeos capturados. Sin embargo, para otro tipo de problemas es posible que se requiera una cámara con mejores prestaciones.

APÉNDICE B

Instalación de Tensorflow-GPU y Keras para la distribución Linux Ubuntu 16.04

Esta instalación puede que ya no sea válida debido a que se realizó en febrero de 2018 y durante este periodo han podido aparecer nuevas actualizaciones. Se realiza en la máquina descrita en el Apéndice A sobre la distribución Linux Ubuntu 16.04.

En primer lugar se realiza la instalación de Anaconda. Se trata de una plataforma que ofrece las principales librerías para Python y R relacionadas con la Ciencia de Datos y el Aprendizaje Automático. La descarga se puede realizar desde el siguiente enlace https://www.anaconda.com/download/#linux.

El siguiente paso es instalar los paquetes que incluye el módulo kernel requeridos por los drivers de NVIDIA:

```
1  $ sudo apt-get install -y linux-image-extra-virtual
2  $ sudo apt-get install linux-source linux-headers-'uname-r'
3  $ sudo reboot
```

Los siguientes pasos a realizar en la instalación se dividen en tres bloques diferentes:

- CUDA
- TensorFlow GPU
- Keras

CUDA

CUDA es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema.[**?**]

La instalación de CUDA es un requisito para poder trabajar con TensorFlow GPU. En la presente instalación se presentaron algunos problemas con Ubuntu 16.04 en el cual se han detectado alguno errores con los drivers de las tarjetas gráficas NVIDIA. La solución encontrada fue la siguiente:

```
1 $ echo -e "blacklist nouveau\nblacklist lbm-nouveau\noptions nouveau modeset=0\
    nalias nouveau off\nalias lbm-nouveau off\n" | sudo tee /etc/modprobe.d/
    blacklist-nouveau.conf
```

```
$ echo options nouveau modeset=0 | sudo tee -a /etc/modprobe.d/nouveau-kms.conf
$ sudo update-initramfs -u
$ sudo reboot
```

Se descarga la versión CUDA 8.0 debido a que las últimas versiones todavía no son estables. Desde el siguiente enlace https://developer.nvidia.com/cuda-toolkit-archive se puede seleccionar la versión y el tipo de instalador. En este caso se ha optado por descargar el fichero con extensión ".run".

Antes de realizar la instalación del fichero descargado se han de desactivar los servicios gráficos. En Ubuntu 16.04 se realiza con la combinación de teclas "ctrl+alt+f1" y escribiendo los siguientes comandos en el terminal que ha aparecido:

```
$ sudo systemctl stop lightdm.service
$ sudo init 3
```

Con los servicios gráficos desactivados ya se puede iniciar la instalación de CUDA y de los drivers compatibles para la gráfica. Si ya se tienen los drivers de la gráfica instalados no es necesario volverlos a instalarlos en este proceso. Por compatibilidad de versiones entre CUDA y cuDNN para que la instalación final funcione correctamente se recomienda instalar la versión 375.82 de los drivers de NVIDIA. Los comando para realizar la instalación son:

```
$ chmod +x cuda_8.0.61_375.26_linux.run
$ sudo sh cuda_8.0.61_375.26_linux.run
```

Una vez finalice la instalación se ha de volver a activar los servicios gráficos:

```
$ sudo modprobe nvidia
$ sudo service lightdm restart
```

Con estos pasos se ha realizado la instalación de CUDA. El siguiente paso es la instalación de la biblioteca cuDNN específica para trabajar con redes neuronales profundas. Para instalar cuDNN se ha de crear una cuenta de desarrollador en NVIDIA (https://developer.nvidia.com/cudnn). Se ha de seleccionar una versión compatible con la versión de CUDA. Para la instalación se ha descargado la versión 6.0 más reciente (*cuDNN v6.0 (April 27, 2017), for CUDA 8.0*).

Una vez descargado el fichero, se ha de descomprimir y copiar los siguientes archivos en el directorio donde se realizó la instalación de CUDA. Los comandos utilizados son:

```
$ tar -xzf cudnn-8.0-linux-x64-v5.1.tgz
$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include/
$ sudo chmod a+r /usr/local/cuda/lib64/libcudnn*
```

Para finalizar la configuración de CUDA y cuDNN se han de añadir las siguientes variables de entornos:

```
$ nano ~/.bashrc
$ export CUDA_HOME="/usr/local/cuda"
$ export LD_LIBRARY_PATH="/usr/local/cuda-8.0/lib64"
$ export PATH="/usr/local/cuda-8.0/bin:$PATH"
$ source ~/.bashrc
```

TensorFlow GPU

Para realizar la instalación de TensorFlow se crea un nuevo entorno en Anaconda donde también se realizará posteriormente la instalación de Keras:

```
$ conda create -n tensorflow_tfg
$ source activate tensorflow_tfg
```

Para instalar la versión correcta de TensorFlow se ha buscar la versión (GPU) compatible con la versión de Python instalada en la máquina. Si se ha instalado Python 3.6 se debe buscar la versión de este y introducir está en el siguiente comando:

```
$ sudo pip install --ignore-installed --upgrade <url_version_tensorflow_gpu>
```

Con esto se finaliza la instalación de TensorFlow y solo queda comprobar que todo se ha instalado correctamente. Comprobación:

```
(tensorflow_tfg)$ python
>>import tensorflow as tf
>>hello = tf.constant('Hello, TensorFlow!')
>>sess = tf.Session()
>>print(sess.run(hello))
```

Keras

Para finalizar se realiza la instalación de Keras dentro del entorno virtual creado anteriormente:

```
(tensorflow)$ pip install keras
```

APÉNDICE C

Detección de objetos con HaarCascade en OpenCV

En este Apéndice se pretende explicar cómo realizar un entrenamiento a partir de los clasificadores HaarCascade que ofrece OpenCV. Paul Viola y Michael Jones en su artículo, *Rapid Object Detection using a Boosted Cascade of Simple Features"* en 2001, propusieron un método para la detección de objetos utilizando clasificadores en cascada basados en características Haar. El entrenamiento se realizar a partir de una gran cantidad de imágenes positivas y negativas.

Muestra negativas

Se han de conseguir una gran cantidad de imágenes donde no aparezca el objeto que se desea detectar. Las muestras negativas se han de almacenar en un directorio específico. Posteriormente se ha de crear un archivo de forma manual donde se almacena la ruta y nombre de cada una de las imágenes. Esta tarea puede ser tediosa si se trabajan con conjuntos grandes de imágenes, es por esto que se ha implementado un *script* Python que automatiza esta última tarea.

```
import cv2
import os

#seleccionar la ruta donde se encuentran las muestras negativas
path="/media/alejandro/TFG/Haar-Cascade/neg/"
dirs = os.listdir( path )
#abrir el fichero donde se escriben las rutas y nombre de las muestras
file = open("negative.txt","w")

#Funci n encargada de recorrer los ficheros de la ruta establecida
# y escribir estas en el fichero de texto
def escribir_negativas():
    for item in dirs:
        print(item)
        file.write("neg/"+item+"\n")

escribir_negativas()
```

Muestras positivas

Al igual que para la muestras negativas también se necesita un gran número de muestras y se han de almacenar en un directorio específico. En cambio se ha de crear un fichero

manual con la ruta y el nombre de la imagen y el número de veces que aparece el objeto en la imagen y las posiciones en que aparece. Un ejemplo puede ser:

```
imagenes/positivas/imagen1.jpg 1 2 3 40 50
```

Después de establecer la ruta, los parámetros establecidos se corresponden con:

- 1: número de objeto en la imagen.
- 2: punto situado en la esquina superior (eje X).
- 3: punto situado en la esquina superior (eje Y).
- 40: ancho de la ventana donde se encuentra el objeto.
- 50: alto de la venta donde se encuentra el objeto.

Si existiera más de un objeto se repetirían los último cuatro parámetros con los valores correspondientes a este. Si existen cerca de mil objetos, esta tarea todavía es más tediosa que la anterior. Para acelerar un poco este proceso se genera otro *script* Python donde a partir de la librería OpenCV se selecciona la ventana o ventanas donde se encuentra un objeto. Después se escriben los datos almacenados en el fichero.

```
import argparse
import cv2
import time
import os
import sys

#flag para saber si se estan marcando los puntos
recortando = False

puntos=[]
#variable para calcular los valores de X,Y,ancho y alto de una ventana
x_min=0
x_max=0
y_min=0
y_max=0
width=0
height=0
coord_list=[]
file = open("positive.txt","w")

def marcar(event, x, y, flags, param):
    global recortando,puntos,width,height,x_max,x_min,y_max,y_min,coord_list
    #Si el boton del mouse esta presionado
    if event == cv2.EVENT_LBUTTONDOWN:
        puntos=[]
        #se almacenan el punto X e Y de donde se ha realizado el click
        puntos.append((x, y))
        x_min=x
        y_min=y
        #se indica que se est  recortando
        recortando = True
    #Cuando se deja de presionar el boton del mouse
    elif event == cv2.EVENT_LBUTTONUP:
        #se almacenan el punto X e Y de donde se ha dejado de realizar el click
        puntos.append((x, y))
        x_max=x
        y_max=y
        #se calcula el ancho de la ventana
```

```
39          width=x_max-x_min
40          #se calcula el alto de la ventana
41          height=y_max-y_min
42          recortando = False
43          #Se a ade una lista con los puntos almacenados en la lista de
                coordenadas
44          coord_list.append((x_min,y_min,width,height))
45          #se dibuja un rectangulo a partir de las coordenadas almacenadas en el
                vector puntos
46          cv2.rectangle(image, puntos[0], puntos[1], (0, 255, 0), 2)
47
48  # se recoge como argumento la ruta donde se encuentran las muestras positivas
49  ap = argparse.ArgumentParser()
50  ap.add_argument("-p", "--path", required=True, help="Directory positive images"
        )
51  args = vars(ap.parse_args())
52  path=args["path"]
53  dirs = os.listdir( path )
54
55  #se recorren todas las imagenes del directorio
56  for item in dirs:
57      #leer imagen
58      image = cv2.imread(path+item)
59      cv2.namedWindow("Imagen")
60      cv2.moveWindow("Imagen", 40,30)
61      #se establece una funci n callback para detectar si se presiona el mouse
62      cv2.setMouseCallback("image", marcar)
63
64      # mostrar imagen hasta que se presione la tecla 'c' o 'q'
65      while True:
66        cv2.imshow("image", image)
67        key = cv2.waitKey(1) & 0xFF
68
69        #si se presiona la tecla 'q' se guarda el fichero con los datos
              almacenados hasta el momento
70        #y se cierra el programa
71        if key == ord("q"):
72          file.close()
73          sys.exit()
74
75        #si se presiona 'c' se indica que se ha finalizado el marcado de una
              imagen y se procede a almacenar
76        #los puntos de esta
77        elif key == ord("c"):
78          #el numero de elementos en la lista de coordenadas indica el numero de
                objetos en la imagene
79          cadena="pos/"+item+" "+ str(len(coord_list))
80          #cada elemento del vector es a su vez una lista con las 2 coordenadas X
                e Y y el ancho y alto
81          for coords in coord_list:
82              cadena=cadena+" "+str(coords[0])+" "+str(coords[1])+" "+str(coords
                    [2])+" "+str(coords[3])
83          #se escribe la informaci n en el fichero de texto
84          file.write(cadena+"\n")
85          #se reinicia la lista para la proxima imagen
86          coord_list=[]
87          break
88  file.close()
```

El siguiente paso después de crear los fichero para las muestra positivas y negativas es crear un fichero de tipo ".vec" con los datos de estos. OpenCV ofrece una funcionalidad para realizar esta tarea, el comando para Ubuntu es el siguiente:

```
$ opencv_createsamples -vec bin_desc.vec -info positive.txt -bg negative.txt -w
    24 -h 24 -num 100
```

Los parámetros que se pasan son:

- -vec: archivo ".vec" donde se almacenan las muestras positivas.
- -info: archivo de descripción de la colección de imágenes marcadas.
- -bg: Archivo de descripción de fondo; contiene una lista de imágenes que se utilizan como fondo para versiones distorsionadas al azar del objeto.
- -w: Ancho (en píxeles) de las muestras de salida.
- -h: Alto (en píxeles) de las muestras de salida.
- -num: número de muestras positivas.

Por último solo queda realizar el entrenamiento. Esta función también se realiza con OpenCV a partir del siguiente comando:

```
opencv_traincascade -data cascade -vec bin_desc.vec -bg negative.txt -numPos
    100 -numNeg 500 -w 24 -h 24 -numStages 17 -maxfalsealarm 0.5
```

Los parámetros que se pasan son:

- -data: directorio donde se almacena el resultado del entrenamiento.
- -vec: archivo ".vec" con las muestras positivas.
- -numPos/numNeg: Número de muestras positivas/negativas utilizadas en el entrenamiento para cada etapa clasificador.
- -w/h: Tamaño de las muestras de entrenamiento (en píxeles)..
- -h: Alto (en píxeles) de las muestras de salida.
- -numStage: número de iteraciones que ha de realizar del clasificador.

Además de los parámetros mencionados existen más para añadir diferentes configuraciones a el clasificador. La información detallada sobre estas se puede encontrar en https://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html.

APÉNDICE D

Implementación para el módulo de entrenamiento y monitorización de la herramienta desarrollada

Este apéndice tiene la finalidad de acercar al lector a la implementación del módulo de entrenamiento y monitorización de la herramienta. Después de leer en el apartado correspondiente cual es la funcionalidad y cómo se interactúa con esta parte, se expone el código utilizado. La estructura del directorio donde se encuentran los ficheros para llevar a cabo esta tarea es :

```
models_tool
├── callbacks.py
├── general.py
├── models
│   ├── VGG16.py
│   └── VGG19.py
├── optimizers
│   └── compile.py
├── results
│   ├── 1528299233638.0.h5
│   └── predict.py
└── topLayers
    └── vgg16_fully.py
```

Antes de exponer el código, se realiza una traza de la ejecución de archivos para el caso de que se trate de un entrenamiento para un modelo preentrenado o para un modelo implementado por el usuario.

Modelo preentrenado

En ambos casos cuando se ejecuta un entrenamiento el primer fichero en ejecutarse es "general.py". Este fichero como se explicó en el apartado recibe una serie de parámetros, donde para este caso recibirá el modelo preentrenado elegido por el usuario. Los modelos preentrenados se encuentra en la "models". Otro parámetro importante es el nombre del fichero que el usuario ha añadido con las capas finales del modelo. A partir del nombre del fichero se accede a este en el directorio "topLayers". Por último solo quedan los parámetros correspondientes al optimizador y la función de error que a su vez son pasados al fichero "compile.py" en el directorio "optmizers".

Modelo implementado por el usuario

Al igual que en el anterior caso primer fichero en ejecutarse es “general.py”. El segundo paso es importar el modelo que se encuentra en la carpeta “models”. Para acceder a este fichero el nombre de este se pasa como parámetros desde la interfaz como el resto de parámetros. Por último solo queda compilar el modelo como en el caso anterior.

Una vez se han completado los entrenamiento para los dos modelos se guarda el modelo resultante en la carpeta “results”. Este modelo servirá para posteriormente realizar pruebas en el módulo correspondiente a las pruebas test y visualización de resultados.

Código implementado

Una vez explicada la estructura y el orden de ejecución de cada uno de los ficheros, se exponen los códigos. El código ha sido comentado para ser más entendible para el lector.

Fichero principal (“general py”):

```
import sys
import os
import numpy as np
import keras
import random
import json
import imp
from keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator, load_img
from keras import models
from keras import layers
from keras import optimizers
import callbacks as c
from keras import backend as K


#carrega d'arguments
data_train = json.loads(sys.argv[1])#datos del modelo de red
data_model = json.loads(sys.argv[2])#datos del conjunto de datos

#tipo de modelo, preentrenado o personalizado
type_learn=data_model["type"]

#carregem moduls, si es un modelo propio no se cargan las capas finales
models = imp.load_source(data_model["model"], os.path.dirname(os.path.abspath(
    __file__))+'/models/'+data_model["model"]+'.py')
modelCompile = imp.load_source("compile", os.path.dirname(os.path.abspath(
    __file__))+'/optimizers/compile.py')
modelTop=""
#si se trata de un modelo preentrenado se cargan las capas finales
if(type_learn != "new"):
  modelTop = imp.load_source(data_model["name_fully"], os.path.dirname(os.path.
      abspath(__file__))+'/topLayer/'+data_model["name_fully"]+'.py')
  #numero de capas a congelar si es un modelo preentrenado
  freeze_layers=int(data_model["freeze_layers"])


#estabLer directorios del conjunto de entrenamiento y validaci n
train_dir=data_train["train_dir"]
validation_dir=data_train["validation_dir"]

#tama o de entrada X e Y
input_x=int(data_model["input_shape_x"])
input_y=int(data_model["input_shape_y"])
dim=int(data_model["dim"])

#nombre para el fichero a guardar
name_file=str(data_train["save_file"])

```

```
#Dependiendo la configuraci n se ha de cambiar el orden de la entrada
if K.image_data_format() == 'channels_first':
  input_s=(dim,input_x,input_y)
  print("FIRST")
else:
  input_s=(input_x,input_y,dim)
  print("LAST")

data_aug=0
#tecnica data augmentation
if(data_aug):
  datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
batch_size = int(data_train["batch_size"])

# Data Generator for Training data
train_generator = datagen.flow_from_directory(
      train_dir,
      target_size=(input_x, input_y),
      batch_size=batch_size,
      class_mode='categorical')

# Data Generator for Validation data
validation_generator = val_datagen.flow_from_directory(
      validation_dir,
      target_size=(input_x, input_y),
      batch_size=batch_size,
      class_mode='categorical',
      shuffle=False)

if(type_learn=="new"):
  #MODELO CREADO POR EL USUARIO
  print("entra")
  model=models.create_model(input_s)
  model=modelCompile.compile(model,data_model)
  #lanzar tarea de entrenamiento, se pasa el numero de iteraciones
  #en el callback tambien se establece el numero de iteraciones para
  #llevar el control desde la interfaz gr fica.
  history = model.fit_generator(
      train_generator,
      steps_per_epoch=train_generator.samples/train_generator.batch_size,
      epochs=int(data_train["epochs"]),
      validation_data=validation_generator,
      validation_steps=train_generator.samples/train_generator.batch_size,
      callbacks=[c.TestCallback(int(data_train["epochs"]))],
      verbose=1)

else:
  #MODELO PARA REDES PREENTRENADAS

  #cargamos el modelo seleccionado
  model=models.create_model(input_s,freeze_layers)
  #se a aden las ultimas capas del modelo
  model=modelTop.topLayers(model)
  #compilar el model
```

```
111   model=modelCompile.compile(model,data_model)
112   model.summary()
113   #lanzar tarea de entrenamiento, se pasa el numero de iteraciones
114   #en el callback tambien se establece el numero de iteraciones para
115   #llevar el control desde la interfaz gr fica.
116   history = model.fit_generator(
117       train_generator,
118       steps_per_epoch=train_generator.samples/train_generator.batch_size,
119       epochs=int(data_train["epochs"]),
120       validation_data=validation_generator,
121       validation_steps=validation_generator.samples/validation_generator.
              batch_size,
122       callbacks=[c.TestCallback(int(data_train["epochs"]))],
123       verbose=1)
124
125   #guardar el modelo con el nombre que se ha recibido desde la interfaz
126   model.save(os.path.dirname(os.path.abspath(__file__))+'/results/'+name_file+"
          .h5")
```

Fichero con modelo preentrendo (“VGG16.py”):

```
1  import keras
2  from keras.applications import VGG16
3  from keras import models
4  from keras import layers
5
6  """
7  Funcion encargada de crear un modelo a partir de un modelo preentrenado
       disponible en la Api
8  de Keras. Parametros:
9    input_s-> dimsniones de entrada
10   freeze-> numero de capas a congelar
11 """
12 def create_model(input_s,freeze):
13   #Se carga el modelo preentrenado VGG16
14   #No se incluyen las ultimas capas
15   vgg_conv = VGG16(weights='imagenet',
16                    include_top=False,
17                    input_shape=input_s)
18
19   #Se congelan el numero de capas establecidas por el usuario
20   for layer in vgg_conv.layers[:10]:
21     layer.trainable = False
22   #Se crea un modelo al cual se a ade el modelo preentrenado
23   model = models.Sequential()
24   model.add(vgg_conv)
25   return model
```

Fichero para compilar modelo (“compile.py”). Para este código solo se introduce un optimizador para ahorrar espacio:

```
1  import keras
2  from keras.utils import to_categorical
3  from keras import models
4  from keras import optimizers
5  from keras import backend as K
6
7  """
8  funci n encargada de compilar un modelo.
9  Par metros:
10     model-> se recibe el modelo de red creado
11     freeze-> en esta funci n se congela el n mero de capas establecidas por
           el usuario cuando creo un modelo de red preentrenada. En caso de
           realizar la compilaci n para un modelo
```

```
    personalizado el valor para el parametro sera 0.
    data_model-> objeto con el optimizador y parametros seleccionados en la
        creacion de un modelo.
"""
def compile(model,data_model):
    if (data_model["select_optimizer"]=="RMSprop"):
        model.compile(optimizer=optimizers.RMSprop(lr=float(data_model["RMSprop
            "]["lr"])),
                      loss=data_model["loss_function"],
                      metrics=['acc'])
    elif (data_model["select_optimizer"]=="SGD"):
    ...

    return model
```

El último código que se expone ("callbacks.py") se encarga de devolver los resultados obtenidos en cada iteración hasta la interfaz de usuario:

```
import zmq
import json
import keras
class TestCallback(keras.callbacks.Callback):
    def __init__(self, test_data):
        self.test_data = test_data

    def on_epoch_end(self, epoch, logs={}):
        epochs = self.test_data
        #loss, acc = self.model.evaluate(x, y, verbose=0)
        context = zmq.Context()
        worker = context.socket(zmq.DEALER)
        worker.setsockopt(zmq.IDENTITY, b'usuari1')
        worker.connect("tcp://localhost:9001")
        dades={"accuracy":logs.get('acc'),"loss":logs.get('loss'),"val_acc":
            logs.get('val_acc'),"val_loss":logs.get('val_loss'),"epoch":epoch}
        js=json.dumps(dades)
        worker.send_string(js)
```

APÉNDICE E

Códigos Python para la detección de objetos

En este Apéndice se adjuntan los códigos implementados para la detección de objetos con ventanas deslizantes. En el primer apartado se mencionan las librerías utilizadas para los códigos.

E.1 Librerías utilizadas

En los dos códigos que se adjuntan en este Apéndice se han importado las siguientes librerías:

```
from keras.models import load_model
from keras.preprocessing.image import img_to_array
import numpy as np
import cv2
```

- En las dos primeras líneas se importan dos módulos de Keras para poder cargar un modelo de red entrenado con Keras y pasar una imagen al formato numpy array.
- Se importa la librería numpy para dar soporte a operaciones con vectores y matrices.
- Para el tratamiento de imágenes se importa la librería cv2 (OpenCV).

E.2 Ventana deslizante horizontal

A continuación se adjunta el código utilizado para la sección donde se explica el algoritmo de ventana deslizante horizontal. El código está comentado para facilitar la comprensión de este al lector.

```
"""
Funcion encargada de predecir la clase de una imagen
Entrada: img: imagen obtenida a partir de una ventana deslizante
         x_init: inicio de la ventana deslizante en el eje X
         x_fin: fin de la venta deslizante en el eje X
         fin_x: punto final en una linea horizontal
Salida:  init: inicio de la pr xima ventana deslizante en el eje X
```

```
8           x_fin: fin de la pr xima ventana deslizante en el eje X
9           fin_x: punto final en una linea horizontal
10  """
11  def procesar(img,x_init,x_fin,fin_x):
12      global pos
13      global primero
14      prediccion = modelo.predict(img)
15
16      #si no se ha encuentra un primero y la probabilidad para la imagen es mayor
            a 0'99 y la clase predecida es la correspondiente al numero 6
17      if(not(primero) and max(prediccion[0])>0.99 and clases[prediccion[0].tolist
            ().index(max(prediccion[0]))]==6):
18          #Se alamcena el numero 6 en la primera posicion del vector numero
19          numero[0]=6
20          #la busqueda se amplia a todo el eje X
21          fin_x=width
22          #se desliza la venta 15px en el eje X que se corresponde con el tama o
                medio de un n mero
23          x_init+=18
24          x_fin+=18
25          #se incrementa una posicion y se marca como que el numero 6 ha sido
                encontrado
26          pos+=1
27          primero=1
28          return x_init,x_fin,fin_x
29      elif(primero and max(prediccion[0])>0.99):
30          #Se alamcena el numero de la clase predecida en la posicion
                correspondiente del vector numero
31          numero[pos]=clases[prediccion[0].tolist().index(max(prediccion[0]))]
32          pos+=1
33          #se realiza un salto de 15px que se corresponde con el tama o medio de
                un n mero
34          x_init+=18
35          x_fin+=18
36          return x_init,x_fin,fin_x
37
38      #si no se ha encontrado ning n numero se desliza la venta 5px en el eje X
39      x_init+=5
40      x_fin+=5
41      return x_init,x_fin,fin_x
42
43
44  # Argumentos de entrada
45  ap = argparse.ArgumentParser()
46  ap.add_argument("-i", "--imagen", required=True,
47      help="ruta de la imagen")
48  ap.add_argument("-m", "--modelo", required=True,
49      help="ruta del modelo")
50  args = vars(ap.parse_args())
51
52  #cargar modelo de la red entrenada
53  modelo = load_model(args["modelo"])
54
55  #etiquetas de las clases
56  clases=[0,5,4,2,9,8,6,7,3,1]
57
58  #vector de 9 posiciones donde se almacena el numero extraido
59  numero=['a']*9
60
61  #flag para indicar si se ha encontrado el numero "6"
62  primero=0
63  #posici n correspondiente a un numero en el vector numero
64  pos=0
65
```

```
66 #cargar imagen sobre la que se aplica la ventana deslizante
67 image = cv2.imread(args["imagen"])
68
69 #reducci n del tama o de la im gen
70 image = cv2.resize(image, (400, 400))
71
72 #eliminar la mitad superior de la im gen
73 height, width, channels= image.shape
74 image = image[int(height/2):height, 0:width]
75
76 #dimensiones de la nueva imagen
77 height, width, channels= image.shape
78
79 """
80 Par metros de la ventana deslizante:
81     x_init: inicio de la ventana deslizante en el eje X
82     x_fin: fin de la venta deslizante en el eje X
83     y_init: inicio de la venta deslizante en el eje Y
84     y_fin: fin de la ventana deslizante en el eje Y
85     fin_x: punto final en una linea horizontal
86     fin_y: indica la  ltima  linea de la imagen
87
88     Ejemplo ventana:
89     (x_init,y_init)------|
90     |                    |
91     |                    |
92     |                    |
93     |                    |
94     |                    |
95     |--------------(x_fin,y_fin)
96
97 """
98 x_init=1
99 x_fin=30
100 y_init=0
101 y_fin=105
102 fin_x =width/2
103 fin_y =height
104
105 "Mientra no se llege al punto final en el eje Y o el numero no se haya
        encontrado"
106 while(y_fin < fin_y and numero[8] == 'a'):
107
108     #imagen obtenida a partir de la ventana deslizante
109     recorte = image[y_init:y_fin, x_init:x_fin]
110     #redimension de la imagen a las dimensiones de la red entrenada
111     recorte = cv2.resize(recorte, (224, 224))
112     recorte = recorte.astype("float") / 255.0
113     recorte = img_to_array(recorte)
114     recorte = np.expand_dims(recorte, axis=0)
115
116     #Llamada a la funci n para procesar la imagen obtenida
117     x_init,x_fin,fin_x=procesar(recorte,x_init,x_fin,fin_x)
118     #Si se ha llegado al final de una linea en el eje X se reestablecen los
            par metos de la ventana
119     if(x_fin>=fin_x):
120         x_init=1
121         x_fin=30
122         y_init+=2
123         y_fin+=2
124         fin_x=width/2
125         primero=0
126         pos=0
```

E.3 Ventana deslizante vertical y horizontal

A continuación se adjunta el código utilizado para la sección donde se explica el algoritmo de ventana deslizante vertical y horizontal. El código está comentado para facilitar la comprensión de este al lector.

```
"""
Funcion encargada de predecir la clase de una imagen
Entrada: img: imagen obtenida a partir de una ventana deslizante
         x_init: inicio de la ventana deslizante en el eje X
         x_fin: fin de la venta deslizante en el eje X
         fin_x: punto final en una linea horizontal
Salida 1(ventana vertical): Y_init: inicio de la pr xima ventana deslizante en el eje Y
                            Y_fin: fin de la pr xima ventana deslizante en el eje Y
                            x_init: inicio de la pr xima ventana deslizante en el eje X
                            x_fin: fin de la pr xima ventana deslizante en el eje X

Salida 2(ventana horizontal): x_init: inicio de la pr xima ventana deslizante en el eje X
                              x_fin: fin de la pr xima ventana deslizante en el eje X
"""
def predictor(img,y_init,y_fin,x_init,x_fin):
    global pos
    global primero
    prediccion = modelo.predict(img)
    if(not(primero) and max(prediccion[0])>0.99 and clases[prediccion[0].tolist().index(max(prediccion[0]))]==6):
        #Se alamcena el numero 6 en la primera posicion del vector numero
        numero[0]=6
        #se incrementa una posicion y se marca como que el numero 6 ha sido encontrado
        pos+=1
        primero=1
        #se incrementa la venta en horizontal y se desliza una distancia de 18 px que se corresponde con el tama o medio de un numero
        x_init+=16
        x_fin+=16
        return y_init,y_fin,x_init,x_fin
    elif(primero and max(prediccion[0])>0.98):
        #Se alamcena el numero de la clase predecida en la posicion correspondiente del vector numero
        numero[pos]=clases[prediccion[0].tolist().index(max(prediccion[0]))]
        x_init+=16
        x_fin+=16
        pos+=1
        return x_init,x_fin

    #si no se ha encontrado el primer 6 se desliza la ventana en vertical
    if(not(primero)):
       #busqueda en vertical
       y_init+=12
       y_fin+=12
       return y_init,y_fin,x_init,x_fin
    else:
       #busqueda en horizontal
       x_init+=10
       x_fin+=10
```

```
47         return x_init,x_fin
48
49
50 # Argumentos de entrada
51 ap = argparse.ArgumentParser()
52 ap.add_argument("-i", "--imagen", required=True,
53     help="ruta de la imagen")
54 ap.add_argument("-m", "--modelo", required=True,
55     help="ruta del modelo")
56 args = vars(ap.parse_args())
57
58
59 #cargar modelo de la red entrenada
60 modelo = load_model(args["modelo"])
61
62 #etiquetas de las clases
63 clases=[0,5,4,2,9,8,6,7,3,1]
64
65 #vector de 9 posiciones donde se almacena el numero extraido
66 numero=['a']*9
67
68 #flag para indicar si se ha encontrado el numero "6"
69 primero=0
70
71 #cargar imagen sobre la que se aplica la ventana deslizante
72 image = cv2.imread(args["imagen"])
73
74 #reducci n del tama o de la im gen
75 image = cv2.resize(image, (400, 400))
76
77 #eliminar la mitad superior de la im gen
78 height, width, channels= image.shape
79 image = image[int(height/2):height, 0:width]
80
81
82 #dimensiones de la nueva imagen
83 height, width, channels= image.shape
84
85 """
86 Parametros de la ventana deslizante:
87     x_init: inicio de la ventana deslizante en el eje X
88     x_fin: fin de la venta deslizante en el eje X
89     y_init: inicio de la venta deslizante en el eje Y
90     y_fin: fin de la ventana deslizante en el eje Y
91     fin_x: punto final en una linea horizontal
92     fin_y: indica la  ltima  linea de la imagen
93
94     Ejemplo ventana:
95     (x_init,y_init)------|
96     |                    |
97     |                    |
98     |                    |
99     |                    |
100    |                    |
101    |--------------(x_fin,y_fin)
102
103 """
104 x_init=1
105 x_fin=30
106 y_init=0
107 y_fin=105
108 fin_x =width/2
109 fin_y =height
110
```

```
111
112 #Mientras no se encuentre el numero de telefono
113 while(numero[8] == 'a'):
114
115     #imagen obtenida a partir de la ventana deslizante
116     recorte = image[y_init:y_fin, x_init:x_fin]
117     #redimension de la imagen a las dimensiones de la red entrenada
118     recorte = cv2.resize(recorte,(224,224))
119     recorte = recorte.astype("float") / 255.0
120     recorte = img_to_array(recorte)
121     recorte = np.expand_dims(recorte, axis=0)
122
123     #Si no se ha encontrado el primer 6 se continua la busqueda en vertical
124     if(not(primero)):
125         #busqueda vertical
126         y_init,y_fin,x_init,x_fin=predictor(recorte,y_init,y_fin,x_init,x_fin)
127     else:
128         #busqueda horizontal
129         x_init,x_fin=predictor(recorte,y_init,y_fin,x_init,x_fin)
130
131     #si se llega al punto establecido como final en el eje Y se restablecen los
          parametros de la ventana deslizante
132     if(y_fin>=fin_y):
133         x_init+=30
134         x_fin+=30
135         y_init=0
136         y_fin=105
137         primero=0
138         pos=0
139
140 print(numero)
```