Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional

Francisco José Núñez Sánchez-Agustino
Máster Universitario en Ingeniería Informática
Área de Inteligencia Artificial

Samir Kanaan Izquierdo
Carles Ventura Royo

01/06/2016

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional*
Nombre del autor:	*Francisco José Núñez Sánchez-Agustino*
Nombre del consultor/a:	*Samir Kanaan Izquierdo*
Nombre del PRA:	*Carles Ventura Royo*
Fecha de entrega (mm/aaaa):	06/2016
Titulación:	*Máster Universitario en Ingeniería Informática*
Área del Trabajo Final:	*Inteligencia Artificial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Convolutional Neural Network*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El presente trabajo es un estudio sobre la aplicación práctica de técnicas de aprendizaje profundo (Deep Learning) en el desarrollo de un sistema de reconocimiento automático de matrículas de vehículos.

Estos sistemas – denominados comúnmente ALPR (Automatic License Plate Recognition) - son capaces de reconocer el contenido de las matrículas de los vehículos a partir de las imágenes capturadas por una cámara fotográfica.

El sistema propuesto en este trabajo se basa en un clasificador de imágenes desarrollado mediante técnicas de aprendizaje supervisado con redes neuronales artificiales convolucionales.

Estas redes son una de las arquitecturas de aprendizaje profundo más populares, y están diseñadas específicamente para resolver problemas de visión artificial, como el reconocimiento de patrones y la clasificación de imágenes.

En este trabajo se estudian también técnicas básicas de procesamiento y segmentación de imágenes - como los filtros de suavizado, la binarización o la detección de contornos - necesarias para que el sistema propuesto pueda extraer el contenido de las matrículas para su posterior análisis y clasificación.

En este documento se demuestra la viabilidad de un sistema ALPR basado en una red neuronal convolucional, constatando la importancia crítica que tiene diseñar una arquitectura de red y un conjunto de datos de entrenamiento adecuados al problema que se quiere resolver.

Abstract (in English, 250 words or less):

This work is a study about the practical application of deep learning techniques for the development of an Automatic License Plate Recognition System.

These systems are able to recognize the plate's characters thanks to the images captured by a camera.

The system proposed in this paper is based on an image classifier developed by supervised learning techniques with convolutional neural networks.

In this paper other basic image processing techniques are also studied, such as image binarization and segmentation, because of their role in the recognition of license plates.

This document shows the viability of an ALPR system based on a convolutional neural network, proving the main importance of adequate network architecture and training dataset based on the problem to be solved.

Agradecimientos

- A mi familia por su apoyo y paciencia durante estos meses.
- A Samir Kanaan Izquierdo por su ayuda y consejos durante este proyecto.

Índice

1. Elección del tema del TFM

El tema escogido para este TFM es el estudio y aplicación de técnicas de aprendizaje profundo (*Deep Learning*) con el objetivo de desarrollar un sistema de visión artificial para el reconocimiento automático de matrículas de automóviles.

Estos sistemas – denominados comúnmente **ALPR** (*Automatic License Plate Recognition*) - son capaces de reconocer el contenido de las matrículas de los vehículos a partir de las imágenes capturadas por una cámara fotográfica.

Sus usos habituales son registrar la entrada y salida de vehículos en un recinto, o como componente de otros dispositivos de control de tráfico, como por ejemplo los cinemómetros o radares de velocidad. En ambos casos, los sistemas ALPR permiten reconocer y digitalizar los números y letras de las matrículas, haciendo posible el cruce de información con una base de datos externa para obtener más datos sobre el vehículo identificado.

El sistema propuesto en este TFM se basa en un clasificador de imágenes desarrollado mediante técnicas de aprendizaje supervisado con **redes neuronales artificiales convolucionales** – también denominadas redes CNN o ConvNet (*Convolutional Neural Network*) -. Estas son una de las arquitecturas de aprendizaje profundo más populares, y están diseñadas específicamente para resolver problemas de visión artificial como el reconocimiento de patrones y la clasificación de imágenes.

2. Objetivos del TFM

El objetivo principal de este trabajo es el estudio y aplicación práctica de las redes neuronales convolucionales (CNN) en un problema de visión artificial como es el reconocimiento óptico de caracteres (OCR) que llevan a cabo los sistema ALPR.

También forman parte de los objetivos de este TFM estudiar y aplicar técnicas básicas de procesamiento y segmentación de imágenes - como los filtros de suavizado, binarización o la detección de contornos - necesarias para poder extraer el contenido de las matrículas para su posterior análisis y clasificación.

Por otro lado, el sistema que se pretende desarrollar debe ser capaz de realizar las siguientes tareas para alcanzar sus objetivos satisfactoriamente:

- Realizar el procesamiento previo de las imágenes capturadas de matrículas eliminando información innecesaria.
- Detectar y extraer los posibles caracteres que componen la matrícula.
- Determinar qué número o letra representa cada uno de los caracteres mediante el modelo generado durante el entrenamiento de la red neuronal.

3. Introducción

3.1 Sistemas de visión artificial

La visión artificial es una rama de la inteligencia artificial cuyo propósito es diseñar sistemas informáticos capaces de "entender" los elementos y características de una escena o imagen del mundo real.

Estos sistemas permiten extraer información – numérica y simbólica - a partir del reconocimiento de objetos y estructuras presentes en la imagen. Para lograrlo llevan a cabo cuatro actividades principales:

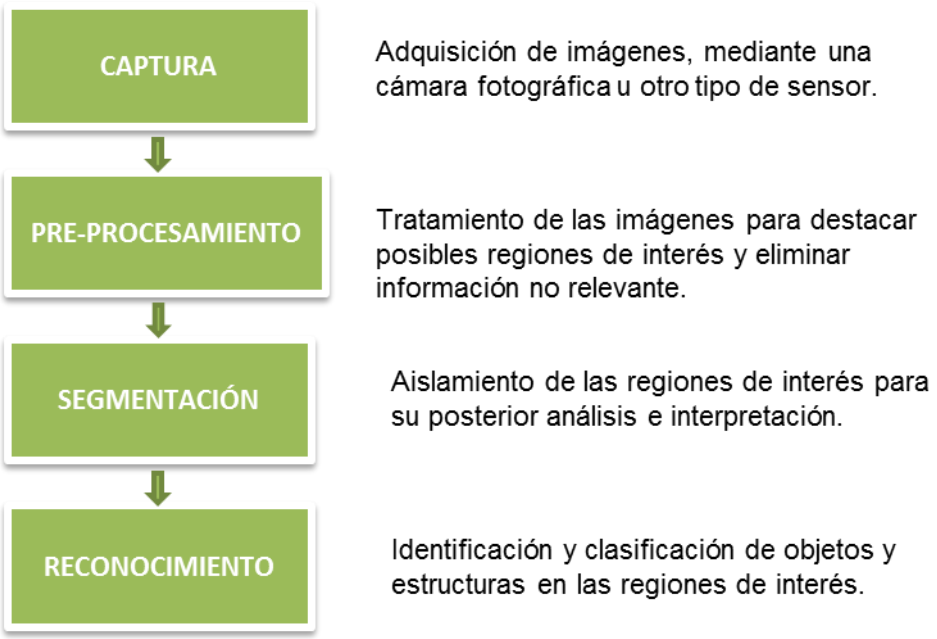

Figura 1 – Sistema de visión artificial

La visión artificial está estrechamente relacionada con las técnicas de procesamiento de imágenes y de reconocimiento de patrones. Las primeras se utilizan para facilitar la localización y detección de áreas de interés en las imágenes; las segundas se emplean para identificar y clasificar los objetos y estructuras detectados en función de sus características.

3.2 Sistemas ALPR

Los sistemas de reconocimiento automático de matrículas de vehículos - **ALPR** (*Automatic License Plate Recognition*) - son un caso particular de los sistemas de visión artificial. Estos sistemas están diseñados para "*leer*" el contenido de las matrículas de los vehículos a partir de las imágenes capturadas por una cámara, y se usan principalmente en dispositivos de vigilancia y control de tráfico.

En estos sistemas el reconocimiento de las imágenes de los números y letras de las matrículas se puede implementar mediante distintas técnicas de aprendizaje de máquina, siendo las más habituales las redes neuronales de tipo *Perceptron Multicapa* (MLP) y las *máquinas de soporte de vectores* (SVM). Incluso algunos sistemas optan por utilizar herramientas comerciales de reconocimiento óptico de caracteres (OCR) para este fin.

3.3 Clasificación de imágenes

El reconocimiento o clasificación de imágenes consiste en asignar a una imagen una etiqueta de un conjunto definido de categorías en función de sus características.

Pese a parecer un problema relativamente trivial desde nuestra perspectiva, se trata de uno de los desafíos más importantes a los que se enfrentan los sistemas de visión artificial. Factores como la escala, las condiciones de iluminación, deformaciones o el ocultamiento parcial de objetos hacen de la clasificación de imágenes una tarea compleja, a la que se ha dedicado un gran esfuerzo para desarrollar sofisticadas técnicas de reconocimiento de patrones, que no siempre producen los resultados esperados.

Desde el punto de vista del aprendizaje de máquina, la clasificación de imágenes es un problema de **aprendizaje supervisado**, en el que el que los algoritmos clasificadores generan un *modelo* a partir de un *dataset* o conjunto de imágenes previamente categorizadas. El modelo obtenido se utiliza posteriormente para clasificar nuevas imágenes.

Para estos algoritmos las imágenes son **matrices** tridimensionales cuyas dimensiones son el ancho, alto y la profundidad de color, siendo el contenido de cada posición de la matriz un valor numérico que representa la intensidad de color de cada píxel de la imagen digital.

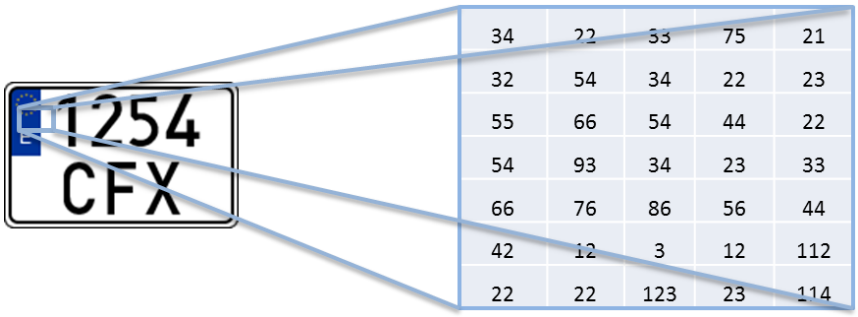

Figura 2- Imagen digital expresada como una matriz (los valores de los píxeles son ficticios)

Hasta la popularización de las redes neuronales convolucionales (CNN), los sistemas clasificadores basados en máquinas de soporte de vectores (**SVM**) eran los que mejores resultados ofrecían en problemas de reconocimiento y clasificación de imágenes.

El principal inconveniente que presentan estos sistemas es que requieren de un proceso previo de **extracción de las características** relevantes de las imágenes, casi siempre diseñado a medida del problema que se pretende resolver, para el que se emplean sofisticadas técnicas de detección de objetos y patrones - histogramas de gradientes orientados (*HOG*), descriptores *SIFT*, etc. - , y en los que suele ser necesaria cierta intuición y conocimiento de los datos de entrada.

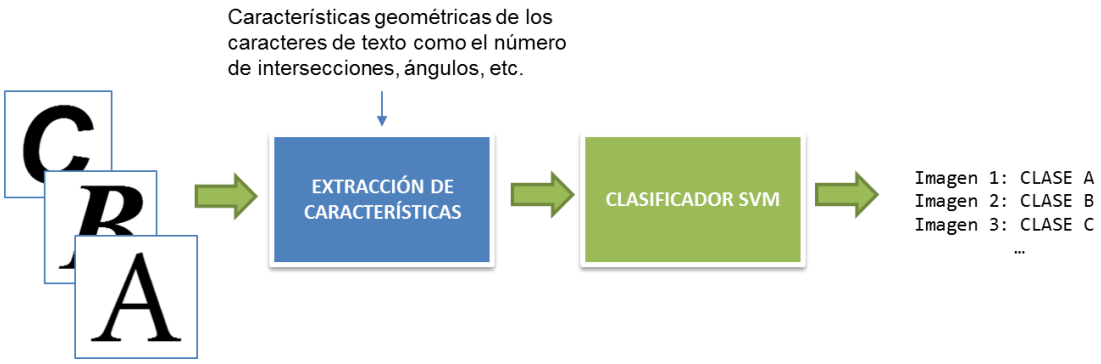

Figura 3 – Sistema de visión artificial basado en un SVM

El algoritmo clasificador SVM se entrena a partir de las características extraídas para un subconjunto del *dataset* de entrenamiento, por lo que la eficacia del modelo generado depende de lo representativas que sean estas.

El aspecto más negativo de estos sistemas es que realmente **no aprenden** las características o atributos relevantes de cada categoría, ya que estas les llegan predefinidas en la etapa de extracción. Además, estos sistemas son extremadamente sensibles a las variaciones de escala, iluminación, perspectiva, etc. que puedan presentar las imágenes.

3.4 Redes neuronales artificiales

Las redes neuronales artificiales son un paradigma de aprendizaje automático inspirado en el funcionamiento del cerebro biológico. Estas redes están compuestas por neuronas interconectadas entre sí que colaboran para producir una salida a partir de los datos de entrada de la red.

Cada neurona artificial o *perceptrón* es una unidad de procesamiento que recibe una serie de señales de entrada que multiplica por un peso determinado (*pesos sinápticos*). La neurona calcula la suma del producto de cada entrada por su peso correspondiente – al que se le suele añadir un factor de corrección o *bias* - , y aplica al valor resultante a una función activación que produce un valor de salida u otro, dependiendo de si la suma de señales y pesos supera un *umbral* determinado.

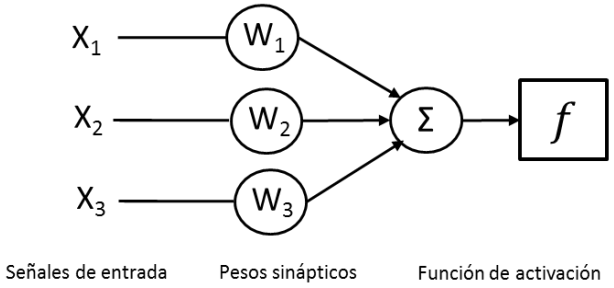

Figura 4 – Esquema de una neurona artificial con tres entradas

Las redes neuronales artificiales se organizan en **capas** de neuronas donde cada capa procesa la información recibida de la anterior. El número de capas y el tipo de función de activación de las neuronas determina la complejidad de los problemas que puede resolver la red: desde detectar patrones sencillos en datos linealmente separables (una capa), a complejas relaciones no lineales entre los datos de entrada (más de tres capas).

Las redes con una o más capas intermedias entre la entrada y la salida son lo que se denominan **redes profundas**, y son la base del *Deep Learning* o aprendizaje profundo. En el siguiente esquema se puede ver una red con una capa oculta, se observa también cómo en estas redes las neuronas de una capa están totalmente interconectadas con las de la siguiente.

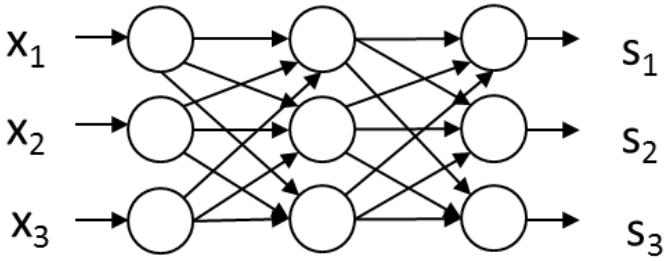

Figura 5 – Red neuronal con una capa oculta o intermedia

El siguiente diagrama muestra de manera simplificada cómo se clasifica una imagen en una red con una capa de tres neuronas que actúan como clasificadores lineales simples:

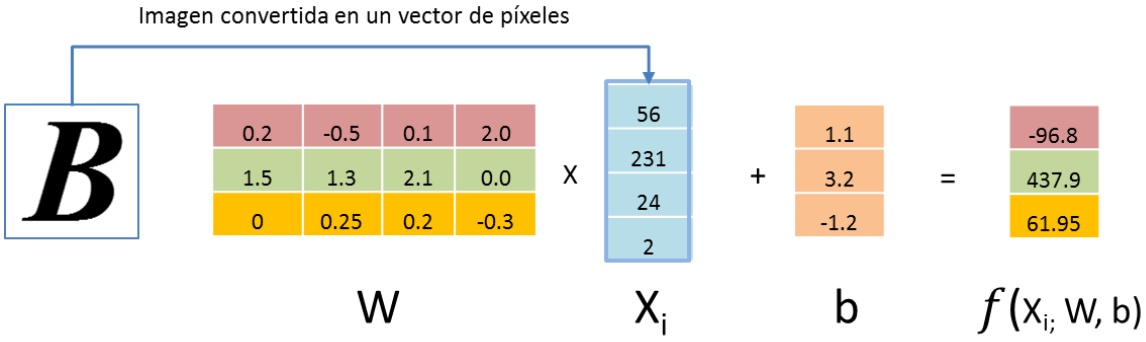

Figura 6 – Clasificación de una imagen en una red neuronal
Fuente: Andrej Karpathy [12]

En la imagen anterior, la matriz *W* son los pesos de las neuronas de la red, X_i es la imagen convertida en un vector de píxeles (solo cuatro para simplificar) y *b* es un vector con los valores de ajuste o *bias*. Cada fila de la matriz *W* representa los pesos de una neurona que a su vez se corresponderían con una clase o categoría de imagen determinada.

El resultado del producto de la matriz *W* por el vector de píxeles X_i más los valores *bias* es un vector con la "puntuación" obtenida por la imagen de entrada para cada una de las posibles clases. En este ejemplo la mayor puntuación sería para la clase sombreada en color verde.

La gran ventaja de las redes neuronales respecto a otros métodos es su mecanismo de **aprendizaje automático**, gracias al cual no es necesario desarrollar procesos de selección y extracción de atributos "*a medida*" como los comentados en el apartado **3.3** para los sistemas basados en SVM.

El algoritmo de aprendizaje de estas redes permite extraer los atributos o características de cada clase a partir de un conjunto de datos de entrenamiento previamente clasificado. Estos atributos son los pesos de las diferentes neuronas de la red, y sus valores se calculan de manera iterativa mediante un método de aprendizaje supervisado denominado ***backpropagation*** o "*propagación de errores hacia atrás*".

A grandes rasgos, el algoritmo consta de dos etapas que se repiten iterativamente por cada elemento del conjunto de entrenamiento:

- En la primera se calcula la clase a la que pertenece el ejemplar de entrada según los valores actuales de los pesos de la red.

  Una vez clasificado, el algoritmo determina la validez de dicha clasificación mediante una **función de error** o coste que calcula lo buena o mala que es, comparándola con la clase a la que realmente pertenece el ejemplo de entrenamiento introducido en la red.

- Conocido el error, la segunda etapa del algoritmo lo propaga hacia atrás a todas las neuronas de la red que han contribuido a la clasificación del ejemplar, recibiendo cada una la "porción" del error correspondiente en función de su aportación, para que actualicen los pesos proporcionalmente, de tal manera que los nuevos valores reduzcan el error de clasificación. El algoritmo empleado para esta optimización suele ser el de **descenso del gradiente**.

La combinación de ambos métodos – propagación hacia atrás y descenso del gradiente – tiene como objetivo minimizar la función de error, actualizando sus parámetros (los pesos de la red) en la dirección opuesta a su *gradiente* (la derivada parcial de la función de error en base a dichos pesos). Este gradiente se calcula en cada una de las capas de neuronas implicadas en la clasificación del ejemplar.

La siguiente imagen describe de manera intuitiva el método de descenso del gradiente con un ejemplo sencillo:

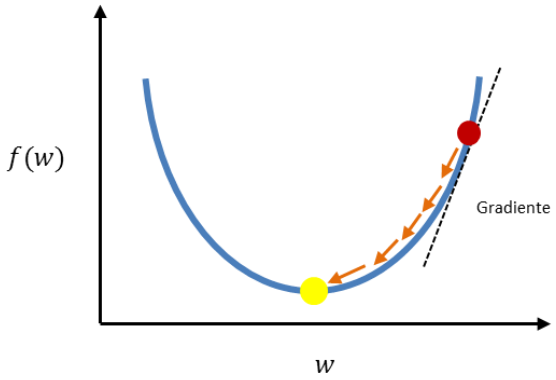

Figura 7 – descenso del gradiente

En la imagen anterior $f(w)$ es la función de error y w un peso determinado. El método del descenso del gradiente permite calcular un valor de w (punto amarillo) mediante el que se obtiene un valor mínimo de la función de error. Para calcular este valor el algoritmo desciende iterativamente por la pendiente (gradiente) desde el valor actual (punto rojo) en el sentido indicado por las flechas, avanzando en cada iteración con una longitud de paso o **tasa de aprendizaje** determinada.

El principal inconveniente de las redes neuronales descritas en este apartado son las dificultades que plantea la interconexión total que existe entre las distintas capas de neuronas cuando aumenta la *dimensionalidad* de los datos de entrada. Por ejemplo: una imagen a color de 300x300 píxeles requiere una capa de entrada con 270.000 pesos (300x300x3), si se añaden varias capas intermedias a la red el número de pesos y *bias* necesarios crecerá desorbitadamente, aumentando el coste computacional y el riesgo de sobre-entrenamiento de la red (*overfitting*).

3.5 Redes neuronales convolucionales

Las **redes neuronales convolucionales** (CNN) son un tipo particular de red neuronal inspirada en el funcionamiento de la corteza visual del cerebro. Estas redes están diseñadas para resolver problemas de visión artificial como el reconocimiento de patrones, aunque pueden tener otros usos como la clasificación de textos o el procesamiento de lenguaje natural.

Al igual que las redes neuronales “convencionales”, reciben una entrada que transforman a través de una serie de capas de neuronas, pero en este caso la entrada es una imagen representada en forma de matriz tridimensional – ancho, alto y profundidad de color - que contiene los valores numéricos de los píxeles. Las capas de neuronas también se organizan de manera tridimensional, y el resultado de las distintas transformaciones llevadas a cabo en la red es la *clase* o categoría a la que pertenece dicha imagen.

Las redes convolucionales se construyen utilizando **cuatro tipos de capas** principales: capa de entrada, capa convolucional, *pooling* y capa totalmente conectada (FC, *Fully Connected*). La estructura de esta última es similar a las redes vistas en el apartado **3.4**.

En las **capas convolucionales** las neuronas no están totalmente interconectadas, sino que lo hacen con una pequeña región de la capa anterior. Además, las neuronas de esta capa comparten los mismos pesos y *bias*, lo que reduce considerablemente el número de parámetros de la red.

De manera gráfica, estas agrupaciones se pueden interpretar como una **ventana** que recorre la imagen o capa de entrada, deslizándose de izquierda a derecha y de arriba a abajo hasta llegar al final, como se puede ver en la siguiente ilustración:

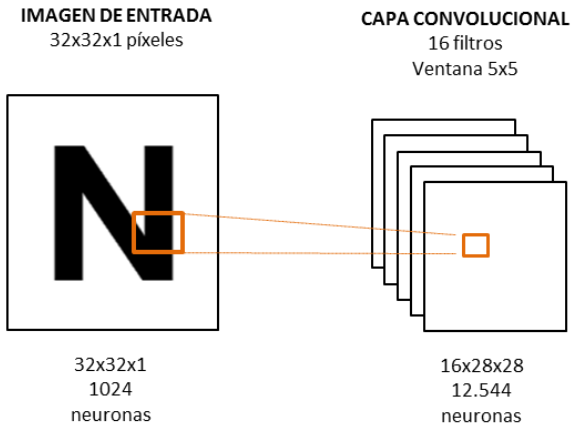

Figura 8 – Ejemplo de capa convolucional

En el ejemplo anterior, la capa de entrada es una imagen en blanco y negro de 32x32 píxeles que contiene una representación de la letra “N”. Las neuronas de la capa convolucional están conectadas a regiones de 5x5 píxeles (rectángulo naranja), es decir, cada neurona está conectada a un total de 25 neuronas de la capa de entrada.

El volumen resultante tiene unas dimensiones de 28x28 píxeles, en este ejemplo este tamaño se calcula a partir de los movimientos que puede realizar la ventana con un desplazamiento de 1 pixel – 27 píxeles a la derecha y 27 píxeles hacia abajo -. El **tamaño de la venta** y su **longitud de desplazamiento** son parámetros que se deben determinar según las características del problema.

A medida que se desplaza la ventana, las neuronas correspondientes de la capa convolucional calculan el producto entre la matriz de pesos compartidos, el *bias* y los valores de los píxeles de la región a la que están conectadas, enviando el resultado de esta operación a una **función de activación** determinada – normalmente la función *ReLU* (*Rectified Linear Unit*), que devuelve el valor *max(0,x)* –.

El resultado de estos cálculos es una serie de **mapas de activación** que permiten detectar la presencia de una característica determinada en la imagen de entrada. Dicha característica viene definida por la matriz de pesos compartida y el valor de *bias*; la combinación de ambos se denomina **filtro** o *kernel* y su aplicación a una región de píxeles se llama *convolución*.

Como se puede ver en la imagen anterior, en las capas convolucionales es habitual que existan varios filtros – matrices de pesos - para poder detectar más de una característica en la imagen. En el ejemplo anterior se han definido 16 filtros para detectar 16 posibles características en la imagen de entrada.

Las capas convolucionales suelen venir acompañadas de una **capa de *pooling*** que condensa la información recogida, dejando sólo los valores máximos de un área determinada de la capa convolucional, lo que permite reducir las dimensiones del volumen de entrada para la siguiente capa.

La **arquitectura** típica de una red CNN es una sucesión de capas convolucionales y *pooling*, siendo habitual que la última capa sea una red neuronal totalmente conectada (FC), que se encarga de calcular las puntuaciones obtenidas por la imagen de entrada para cada una de las clases o categorías definidas en el problema.

El **algoritmo de aprendizaje** de estas redes es el visto en el apartado **3.4**: *backpropagation* + descenso del gradiente u otro método similar. Pero en estas redes el entrenamiento tiene como objetivo que las distintas capas de neuronas aprendan los filtros o características de bajo y alto nivel (líneas, aristas, etc.) que representan a cada clase o categoría de imagen del problema.

Las redes CNN son hoy por hoy el "*estado del arte*" en el campo de la visión artificial, donde han demostrado una eficacia muy superior a otras técnicas, y esto es debido a tres **ventajas** clave de su arquitectura:

1. Posibilidad de detectar un atributo en cualquier posición de la imagen gracias al deslizamiento de la ventana.
2. Tolerancia a leves variaciones de rotación, traslación o escala, gracias a las capas de *pooling*.
3. Capacidad para reconocer atributos o características complejas de alto nivel mediante la combinación de los filtros de las capas convolucionales.

Un ejemplo del predominio de las redes convolucionales son los resultados de las prestigiosas pruebas *ImageNet* **[1]**. En estas pruebas anuales compiten equipos de expertos en visión artificial de todo el planeta presentando sistemas que intentan clasificar imágenes en 200 categorías predefinidas, partiendo de un conjunto de entrenamiento compuesto por más de 450.000 imágenes.

Desde el año 2012 hasta la fecha son los sistemas basados en redes CNN los que obtienen los mejores resultados, con tasas de error considerablemente más bajas que las ofrecidas por otras técnicas de inteligencia artificial, como se puede comprobar en la página de resultados de 2015 **[2]**.

4. Sistema propuesto

El siguiente esquema resume el funcionamiento del sistema de visión artificial propuesto para el reconocimiento automático de matrículas de automóviles:

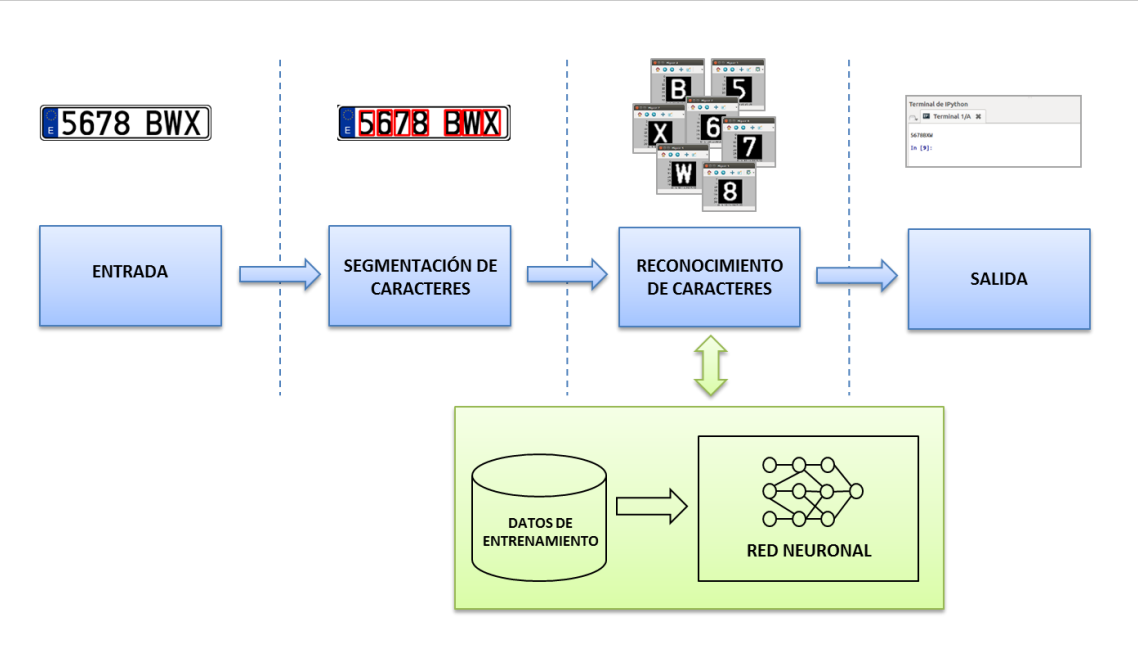

Figura 9 – Esquema del sistema ALPR propuesto

Como se puede observar en el diagrama, la **entrada** del sistema son imágenes de matrículas de vehículos. En este TFM se utilizarán tanto imágenes reales como generadas por ordenador, escaladas automáticamente a una resolución aproximada de 230x50 píxeles para facilitar su procesamiento.

Una vez introducida una imagen, el sistema ejecutará el proceso de **segmentación** que consiste en la detección de los posibles números y letras presentes en la matrícula. El módulo de código que realiza este proceso lleva a cabo una segmentación basada en la detección de contornos, empleando para ello las funciones de la popular librería de visión artificial *OpenCV*.

La etapa de **reconocimiento de caracteres** se realizará mediante una red neuronal convolucional (CNN) entrenada previamente con un conjunto de datos con imágenes de números y letras generadas por ordenador con distintas tipografías y estilos.

La **salida** del sistema será una cadena de texto compuesta por los caracteres detectados en la imagen de la matrícula de entrada.

5. Planificación

A continuación se resume la planificación de este TFM mediante una tabla de tareas a realizar, la relación de entregables y un diagrama de Gantt.

5.1 Tareas

Tarea	Duración	Inicio	Fin
Plan de trabajo	15 días	01/03/16	15/03/16
Selección del tema	8 días	01/03/16	08/03/16
Elaboración de la propuesta	5 días	09/03/16	13/03/16
Elaboración del plan de trabajo	2 días	14/03/16	15/03/16
Preparación	16 días	16/03/16	31/03/16
Ampliación de conocimientos sobre redes neuronales y visión artificial	7 días	16/03/16	22/03/16
Adquisición de conocimientos básicos sobre manipulación de imágenes	3 días	23/03/16	25/03/16
Análisis y selección de los algoritmos y herramientas disponibles	2 días	26/03/16	27/03/16
Análisis y selección de los conjuntos de datos necesarios	4 días	28/03/16	31/03/16
Ejecución	42 días	01/04/16	12/05/16
Tratamiento y transformación de los conjuntos de datos de entrenamiento y prueba	10 días	01/04/16	10/04/16
Diseño y codificación de la red neuronal	10 días	11/04/16	20/04/16
Entrenamiento y optimización de le red neuronal	10 días	21/04/16	30/04/16
Diseño y codificación del subsistema de tratamiento de imágenes	8 días	01/05/16	08/05/16
Ejecución del sistema de clasificación con datos reales	4 días	09/05/16	12/05/16
Análisis de resultados y conclusiones	13 días	13/05/16	25/05/16
Preparación de la presentación y defensa del TFM	7 días	26/05/16	01/06/16
Redacción de la memoria	73 días	21/03/16	01/06/16
TOTAL	**93 días**	**01/03/16**	**01/06/16**

5.2 Entregables

Entregable	Descripción	Fecha
PEC1	Descripción del tema del TFM, objetivos, estado del arte, herramientas elegidas y plan de trabajo.	09/03/16
PEC2	Índice con la estructura completa de la memoria y contenido del apartado de planificación.	06/04/16
PEC3	50% apartados de la memoria redactados, breve informe de resultados obtenidos en los experimentos.	04/05/16
PEC4	Versión completa de la memoria con resultados y conclusiones. Presentación del trabajo en vídeo.	01/06/16

5.3 Diagrama de Gantt

Nombre de tarea	Duración	Comienzo	Fin
TFM	**93 días**	**mar 01/03/16**	**mié 01/06/16**
Plan de trabajo	**15 días**	**mar 01/03/16**	**mar 15/03/16**
Selección del tema	8 días	mar 01/03/16	mar 08/03/16
Elaboración de la propuesta	5 días	mié 09/03/16	dom 13/03/16
Elaboración del plan de trabajo	2 días	lun 14/03/16	mar 15/03/16
Preparación	**16 días**	**mié 16/03/16**	**jue 31/03/16**
Ampliación de conocimientos sobre redes neuronales y visión artificial	7 días	mié 16/03/16	mar 22/03/16
Adquisición de conocimientos básicos sobre manipulación de imágenes	3 días	mié 23/03/16	vie 25/03/16
Análisis y selección de los algoritmos y herramientas disponibles	2 días	sáb 26/03/16	dom 27/03/16
Análisis y selección de los conjuntos de datos necesarios	4 días	lun 28/03/16	jue 31/03/16
Ejecución	**42 días**	**vie 01/04/16**	**jue 12/05/16**
Tratamiento y transformación de los conjuntos de datos de entrenamiento y prueba	10 días	vie 01/04/16	dom 10/04/16
Diseño y codificación de la red neuronal	10 días	lun 11/04/16	mié 20/04/16
Entrenamiento y optimización de le red neuronal	10 días	jue 21/04/16	sáb 30/04/16
Diseño y codificación del subsistema de tratamiento de imágenes	8 días	dom 01/05/16	dom 08/05/16
Ejecución del sistema de clasificación con datos reales	4 días	lun 09/05/16	jue 12/05/16
Análisis de resultados y conclusiones	**13 días**	**vie 13/05/16**	**mié 25/05/16**
Preparación de la presentación y defensa del TFM	**7 días**	**jue 26/05/16**	**mié 01/06/16**
Redacción de la memoria	**73 días**	**lun 21/03/16**	**mié 01/06/16**

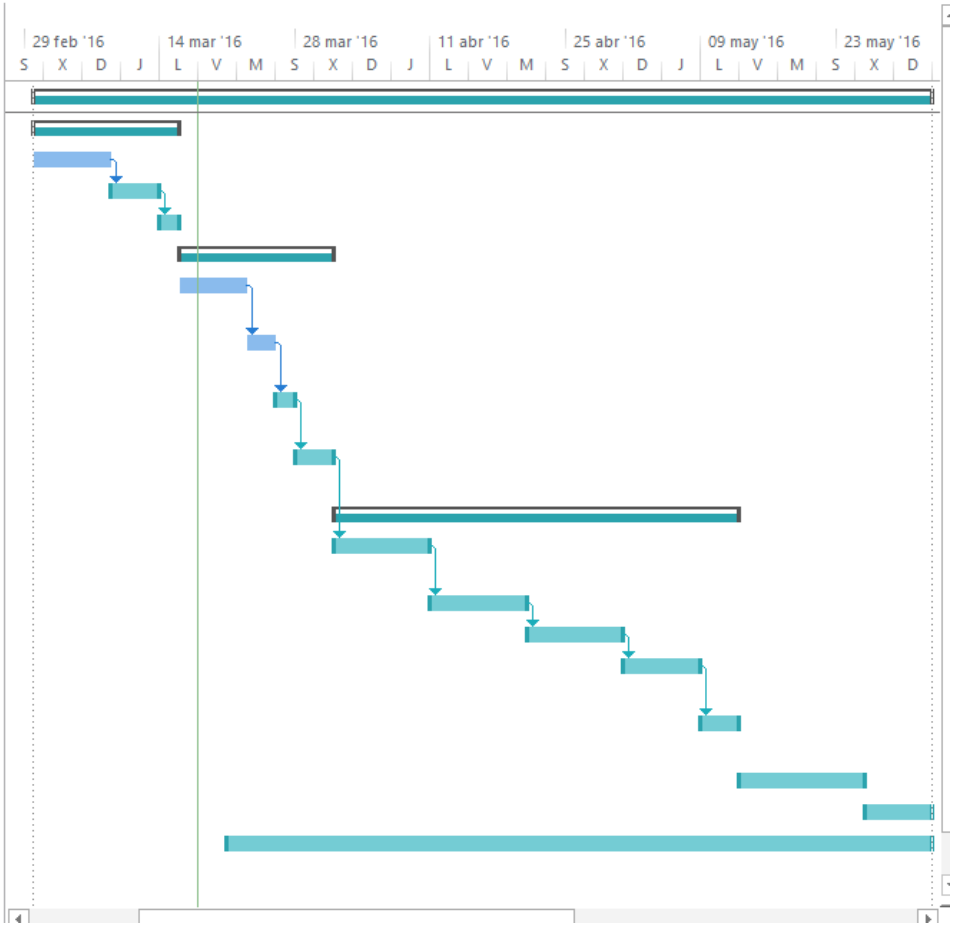

Figura 10 – Diagrama de Gantt de la planificación del TFM

6. Descripción de los datos

6.1 Conjunto de datos de entrenamiento

La eficacia de un sistema de clasificación de imágenes basado en una red neuronal convolucional depende tanto de la arquitectura y ajustes de la red, como del conjunto de datos empleado para su entrenamiento. Este último debe ser suficientemente amplio y representativo del problema que se quiere resolver.

Para entrenar la red neuronal del sistema propuesto se ha elegido el conjunto de datos *Chars74k* creado por T. de Campos y M. Varma de Microsoft Research India [3]. Este *dataset* cuenta con más de 74.000 imágenes de números y letras en formato PNG organizadas en tres colecciones: caracteres manuscritos, caracteres extraídos de fotografías de escenas cotidianas y caracteres generados por ordenador.

Esta última colección cuenta con más de 60.000 imágenes en escala de grises de dígitos y letras representados con diferentes fuentes y estilos (normal, negrita y cursiva), a priori idóneas para que la red CNN "aprenda" a reconocer los patrones de los distintos números y letras de las matrículas.

Figura 11 – Muestra de imágenes del *dataset* Chars74

Sin embargo, no todas las imágenes de esta colección son válidas para el entrenamiento de la red: las letras en minúscula y los ejemplares generados mediante fuentes "exóticas" no son representativos del problema planteado. Así por ejemplo, la siguiente ilustración muestra algunas imágenes que han sido descartadas según este criterio.

Figura 12 – Algunos ejemplares descartados (representan las letras "H", "A", "D" y "W" respectivamente)

Excluyendo este tipo de imágenes la colección de caracteres generados por ordenador que se utilizará en el entrenamiento queda reducida a un total de 34.584 imágenes de los dígitos de 0-9 y letras de A-Z, con una media aproximada de 950 imágenes por cada clase de número y letra.

6.2 Conjunto de datos para validar el sistema

Obtener un *dataset* con imágenes reales de matrículas de vehículos es una tarea más complicada de lo que pudiera parecer, ya que en muchas legislaciones esta información es considerada como un dato de carácter personal.

En el caso de España, un informe jurídico de la Agencia Española de Protección de Datos (AEPD) del año 2006 [4] concluyó que: "dado que no se requiere un esfuerzo desproporcionado para identificar al propietario del vehículo, el simple número de la matrícula puede ser considerado un dato de carácter personal; sujeto por tanto a lo dispuesto en la Ley Orgánica 15/1999 de Protección de Datos de Carácter Personal (LOPD)".

Para evitar este problema se ha optado por validar el sistema ALPR con un conjunto de imágenes de matrículas de vehículos en su mayoría generadas por ordenador, incluyendo también alguna fotografía real tomada con el consentimiento del propietario del vehículo.

Estas matrículas "sintéticas" se han obtenido mediante las herramientas gratuitas de los sitios Web *platesmania.com* [5] y *acme.com* [6].

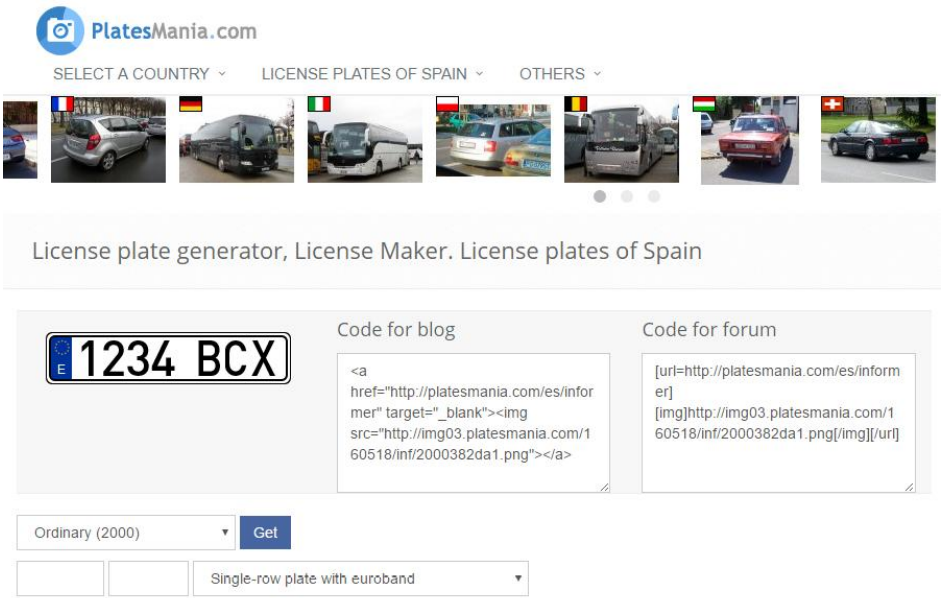

Figura 12 – Generador de matrículas del sitio Web platesmania.com

La primera permite crear matrículas según el formato nacional vigente: dos grupos de caracteres constituidos por un número de cuatro cifras, desde el 0000 al 9999, y tres letras, empezando por las letras BBB y terminando por las letras ZZZ, en donde se suprimen las cinco vocales y las letras Ñ, Q, CH y LL.

Por otro lado, la herramienta del sitio Web *acme.com* permite generar matrículas según distintos formatos estatales de Estados Unidos, pero sin ninguna restricción a la hora de introducir combinaciones de letras y números.

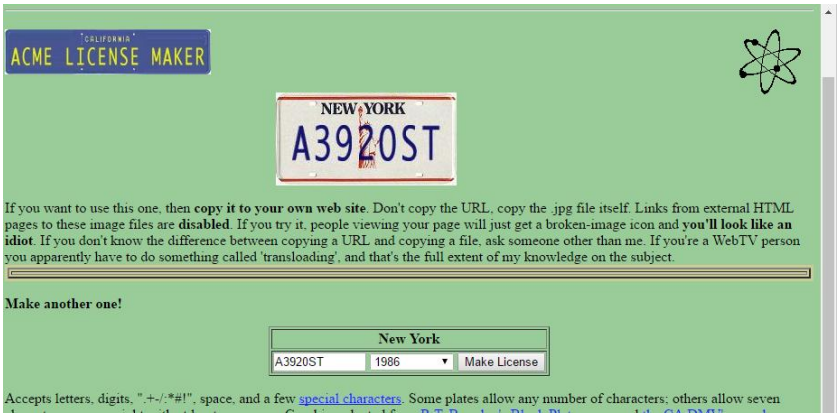

Figura 13 – Generador de matrículas de acme.com

A continuación se muestran algunos ejemplos de matrículas "sintéticas" generadas con estas herramientas para validar el sistema:

Figura 14 – Ejemplos de matrículas generadas por ordenador

7. Tecnologías empleadas

7.1 Herramientas

Las principales herramientas utilizadas para desarrollar el sistema ALPR han sido el lenguaje de programación Python y las librerías *TensorFlow* y *OpenCV*, estas últimas empleadas para generar la red convolucional y el sistema de pre-procesamiento de las imágenes de las matrículas respectivamente.

7.1.1 Python

Python es un lenguaje de programación interpretado de propósito general, multiplataforma y multiparadigma (imperativo, orientado a objetos y funcional). Se trata de un lenguaje fuertemente tipado, es decir, no se puede asignar a una variable un valor de tipo distinto al inicial sin una conversión explícita, siendo a su vez este tipado *dinámico*: el tipo de las variables se determina en tiempo de ejecución en función del valor asignado.

Los motivos de la elección de este lenguaje es una sintaxis clara e intuitiva que facilita la legibilidad del código, y la gran cantidad de librerías especializadas disponibles, como por ejemplo *NumPy*, que permite realizar de manera sencilla todo tipo de operaciones con vectores y matrices, fundamentales a la hora de trabajar con imágenes y redes neuronales artificiales.

7.1.2 OpenCV

OpenCV es una librería de visión artificial desarrollada por Intel y actualmente publicada bajo licencia BSD, que cuenta con más de medio millar de algoritmos optimizados para realizar las principales tareas de visión artificial, como el procesamiento de imágenes, detección de características o reconocimiento de objetos. La librería cuenta además con diferentes algoritmos de aprendizaje de máquina como máquinas de soporte de vectores (SVM), Naïve Bayes o KNN entre otros.

Está escrita en C++, es multiplataforma y cuenta con interfaces para trabajar con lenguajes como Java o Python. La gran cantidad de algoritmos disponibles, su velocidad de ejecución y la extensa comunidad de usuarios de que dispone, hacen de *OpenCV* una herramienta indispensable para desarrollar sistemas de visión artificial basados en software de código abierto.

7.1.3 TensorFlow

Se trata de una librería de código abierto desarrollada por Google para facilitar el diseño, construcción y entrenamiento de sistemas de aprendizaje profundo. *TensorFlow* se caracteriza por un modelo de programación en el que tanto las operaciones como los datos se almacenan internamente en una estructura en grafo, lo que facilita la visualización de las dependencias entre operaciones y su asignación a diferentes dispositivos como los procesadores gráficos.

Si bien es cierto que existen otras librerías *Deep Learning* compatibles con Python que ofrecen un modelo de programación similar; TensorFlow presenta una sintaxis bastante clara y cuenta con una potente herramienta de visualización (*TensorBoard*) que permite entender, depurar y optimizar el grafo de computación ofreciendo todo tipo de estadísticas.

Además, detrás de TensorFlow se encuentra el equipo de investigación del proyecto *Google Brain*, que cuenta con algunos de los principales expertos en *Deep Learning* como Geoffrey Hinton y Alex Krizhevsky, autores del primer sistema basado en una red CNN que ganó la prueba ImageNet en 2012 [7].

7.2 Tratamiento de imágenes

Tal y como se comentaba en la descripción del sistema del apartado **4**, el primer proceso que se ejecuta una vez introducida la imagen de una matrícula es la segmentación de caracteres, cuyo objetivo es detectar aquellas regiones de la imagen que contienen los números y letras que posteriormente tratará de identificar el clasificador de la red neuronal.

El módulo Python que ejecuta este proceso está basado en *OpenCV*, y emplea las siguientes técnicas de procesamiento de imágenes:

7.2.1 Conversión a escala de grises

La información de color no es necesaria para el problema planteado y su eliminación facilita la detección de los caracteres de la matrícula. Por este motivo, la primera transformación que se le aplica a la imagen es una conversión a escala de grises, o lo que es lo mismo, convertir la imagen en una matriz en la que cada valor representa el nivel de gris del pixel correspondiente. Esta transformación se lleva a cabo mediante la función *cvtColor* de *OpenCV*.

7.2.2 Desenfoque Gaussiano

La segunda transformación que se realiza a la imagen es aplicar un efecto de suavizado para eliminar el ruido. En concreto, se aplica un filtro de desenfoque Gaussiano que consiste en mezclar ligeramente los niveles de gris entre píxeles adyacentes, lo que da como resultado una imagen con los bordes más suaves en la que se pierden pequeños detalles, de manera similar a lo que ocurre en las fotografías desenfocadas. Esta transformación se realiza mediante la función *GaussianBlur* de *OpenCV*.

7.2.3 Thresholding

El *thresholding* o “técnica del valor umbral” es un método que permite convertir una imagen en blanco y negro estableciendo un valor a partir del cual todos los píxeles que lo superen se transforman en un color binario (blanco o negro), y el resto en el contrario. En el caso de las matrículas de vehículos esta transformación permite obtener una imagen en la que quedan claramente definidos los contornos de los caracteres, facilitando el proceso de segmentación o aislamiento de las áreas que los contienen.

En el sistema propuesto esta transformación se realiza mediante la función *adaptiveThreshold* de OpenCV, en la que el valor umbral se calcula para distintas regiones de la imagen, ofreciendo buenos resultados incluso si existen variaciones de iluminación en la imagen.

7.2.4 Detección de contornos

Los contornos o bordes son curvas que unen conjuntos de píxeles contiguos que tienen el mismo color o intensidad. Estas curvas permiten localizar las fronteras de los objetos de la imagen, y su detección es fundamental para que un sistema de visión artificial pueda reconocer o detectar formas en una imagen.

En el sistema propuesto esta detección es la última etapa de pre-procesamiento de la imagen de la matrícula, y se lleva a cabo mediante las funciones *findContours* y *boundingRect* de *OpenCV*. La primera toma como entrada la imagen binaria de la fase anterior y genera una serie de vectores de puntos por cada contorno detectado. La segunda se utiliza para dibujar un rectángulo que rodea al contorno, lo que facilitará la posterior extracción las regiones de la imagen que contienen los números y letras de la matrícula.

En la siguiente ilustración se puede comprobar el resultado de las cuatro técnicas aplicadas a la imagen de una matrícula:

Figura 15 – Tratamiento de la imagen de entrada

7.3 Arquitectura de la red neuronal

El componente principal del sistema ALPR presentado en este trabajo es su red neuronal convolucional (CNN). Esta es la encargada de clasificar los caracteres de las matriculas a partir de las imágenes extraídas en el módulo de segmentación, para lo cual es necesario un proceso previo de entrenamiento para que la red aprenda las características que definen a las distintas letras y números que pueden estar presentes en una matrícula.

El siguiente esquema resume la arquitectura de la red convolucional del sistema:

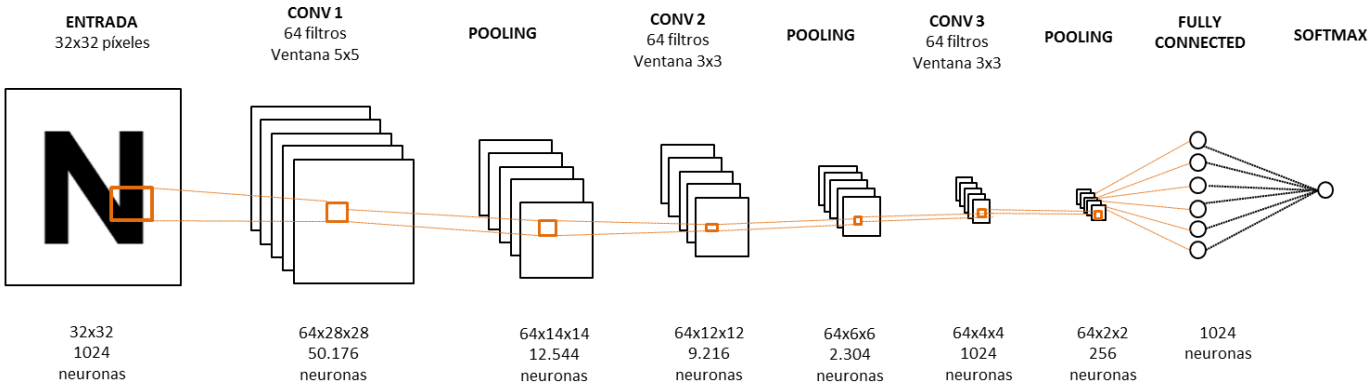

Figura 16 – Arquitectura de la red neuronal convolucional propuesta

En el esquema anterior se puede ver que como la **entrada** de la red recibe una imagen de 32x32 píxeles, lo que significa que serán necesarias un total de 1.024 neuronas para la primera capa. A continuación se encuentran tres **capas convolucionales** - *CONV1*, *CONV2* y *CONV3* - con funciones de activación ReLU, acompañadas cada una por una capa de *pooling*.

Las neuronas de la primera capa convolucional (CONV1) están conectadas a regiones de 25 neuronas de la entrada mediante una ventana de 5x5 píxeles. Esta ventana se desplaza horizontalmente por la imagen con un tamaño de paso o *stride* de 1 píxel y sin relleno a ceros (*padding 0*), para detectar la presencia de determinadas características en la imagen entrada.

En esta capa hay definidos un total de 64 filtros – representados por su correspondiente matriz de 25 pesos y valor de *bias* - que permiten identificar en esta capa un total de 64 características diferentes para clasificar los distintos tipos de dígitos y letras de las matrículas.

En el esquema se puede observar como el resultado de esta primera capa de convolución son 64 volúmenes de 28x28 píxeles (50.176 neuronas) que, una vez aplicada la capa de *pooling* se condensan en otros 64 volúmenes de 14x14 píxeles (9.216 neuronas). Esto es debido al tamaño de bloque de 2x2 píxeles elegido para la capa de *pooling*, que reduce las dimensiones de cada uno de los 64 volúmenes a la mitad (de 28x28 a 14x14).

La segunda y tercera capa convolucional – CONV2 y CONV3 - presentan una estructura similar salvo por el tamaño de ventana, que se reduce a 3x3 píxeles.

Estas tres capas convolucionales constituyen lo que se denomina en la literatura de redes CNN un "*sistema de extracción de características entrenable*", ya que durante el proceso de entrenamiento la red aprende los pesos y *bias* de las características de alto y bajo nivel que definen a cada una de las 36 distintas clases posibles de números y letras (dígitos de 0 a 9 y letras de A-Z).

Finalmente, las últimas capas de la red CNN están formadas por una capa de 1024 neuronas totalmente conectadas (FC) seguida de una capa que ejecuta la función *softmax* para calcular, a partir de las características extraídas en las capas convolucionales, la probabilidad de que la imagen pertenezca a cada una de las 36 clases de caracteres posibles.

Esta arquitectura está basada en la red convolucional *LeNet-5* propuesta por Yann Lecun **[8]** para el reconocimiento de dígitos, adaptando el número de capas y los valores de los distintos parámetros a las necesidades del problema planteado en este trabajo.

Este diseño también sigue la estrategia general planteada por Simard et al. **[9]** para el análisis visual de documentos mediante redes convolucionales, que consiste extraer características sencillas de los caracteres en las primeras capas de la red, para posteriormente convertirlas en características complejas gracias a la combinación de los distintos filtros de las sucesivas capas convolucionales.

8. Ejecución

8.1 Tratamiento de los datos

A continuación se describen las distintas operaciones de pre-procesamiento que se llevan a cabo en el sistema, tanto en el módulo de segmentación de caracteres, como durante el proceso de carga del conjunto de imágenes de entrenamiento de la red neuronal.

8.1.1 Pre-procesamiento de las imágenes de entrada

En el apartado 7.2 se describieron las distintas técnicas de tratamiento de imágenes que se aplican a las matrículas que llegan al sistema - conversión a escala de grises, desenfoque Gaussiano, *thresholding* y detección de contornos -, con el fin de aislar aquellas regiones que pueden contener un número o una letra de la matrícula.

Una vez identificadas estas "regiones de interés" (ROI), el módulo Python encargado del proceso de segmentación – página **43** - realiza una serie de **comprobaciones** para determinar si su contenido es efectivamente un dígito o una letra que deba ser analizado y clasificado por la red neuronal.

Estas comprobaciones consisten básicamente en verificar si la altura, anchura y la relación de aspecto en píxeles - es decir, la relación entre ancho y alto de la región de interés - se encuentran dentro de unos valores determinados. Los valores mínimos y máximos de estos parámetros vienen determinados por las proporciones que deberían tener los caracteres de una matrícula estándar de aproximadamente 230x50 píxeles, siendo esta la resolución a la que se escalan todas las imágenes que entran en el sistema, tal y como se indica en el apartado 4.

Si las proporciones de una región determinada son válidas, es decir, están dentro de los límites, el siguiente paso consiste en extraer su contenido y guardarlo como una imagen de 32x32 píxeles que pueda ser analizada por la red CNN del sistema. De manera resumida, esta transformación se realiza mediante las siguientes acciones:

1. Se extrae de la imagen de la matrícula en escala de grises la región delimitada por el rectángulo generado mediante *boundingRect* presentada en el apartado **7.2.4**.
2. El contenido de la región se redimensiona para que tenga una altura máxima de 28 píxeles, manteniendo la relación de aspecto original.

3. La imagen resultante se convierte en binaria (blanco y negro) y se invierte mediante la función *threshold* indicada en el apartado **7.2.3**, de tal forma que los píxeles del color de fondo tengan valor 0 (color negro).
4. A continuación se añade un borde negro que rodea a la imagen mediante la función *copyMakeBorder* de *OpenCV*. Este borde se genera con las dimensiones adecuadas para que el dígito o letra contenido en la región aparezca centrado.
5. La imagen resultante se redimensiona a 32x32 píxeles.

En la siguiente imagen se puede ver el resultado de las acciones anteriores:

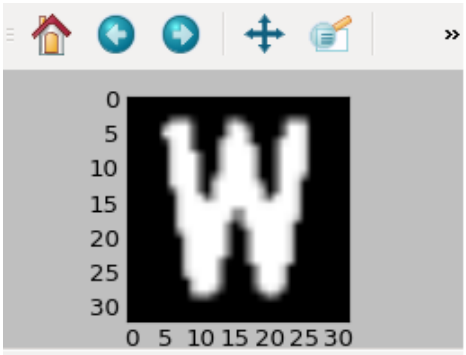

Figura 17 – Resultado de procesar el contenido de una región de interés

El último paso de este de proceso consiste en convertir la imagen del contenido de la región de interés en un vector de píxeles de 1024 elementos, y almacenarlo en una matriz junto con los otros posibles números y letras del resto de regiones de interés detectadas en la imagen.

Finalmente, las filas de esta matriz se ordenan por el valor de la coordenada *x* de la región de a la que pertenecen, para poder así mantener el mismo orden en el que aparecieron en la imagen original de la matrícula.

8.1.2 Pre-procesamiento del conjunto de datos de entrenamiento

Para poder entrenar la red CNN es necesario importar el subconjunto de imágenes representativas del problema que contiene el *dataset Chars74K* en una matriz de datos. Tal y como se comentó en el apartado **6.1**, este subconjunto del *dataset* está compuesto por imágenes en escala de grises de caracteres generados por ordenador correspondientes a los números de 0-9 y letras mayúsculas de A-Z.

Como se indicaba en el apartado **3.4**, la gran ventaja de utilizar una red neuronal como algoritmo de clasificación es su capacidad de aprendizaje automático, gracias al cual no es necesario desarrollar procesos de extracción de atributos ni emplear técnicas para reducir la dimensionalidad como PCA (*Principal Component Analysis*) o similares; lo que simplifica enormemente las tareas de pre-procesamiento de los datos de entrenamiento.

En concreto, el módulo de código Python encargado de importar las imágenes de entrenamiento – página **42** - realiza las siguientes acciones:

1. Recorre las 36 carpetas de imágenes (una por cada clase de dígito o letra) de la colección, y dentro de cada carpeta redimensiona cada imagen de 128x128 píxeles a 32x32 píxeles, que es la resolución válida para la entrada de la red CNN del sistema.
2. Cada imagen se invierte de tal manera que el valor de los píxeles del color de fondo valgan 0 (color negro) y los que representan al contorno de la letra o dígito sean positivos.
3. La imagen invertida se transforma en un vector de 1.024 posiciones (32x32 píxeles), y este a su vez se añade a una matriz denominada "*X"* en la que se almacenan todos los vectores de imágenes procesadas.
4. Por cada imagen guardada en la matriz se crea una entrada en un vector denominado "*y",* que almacena para cada imagen un número de 0 a 36 que identifica la clase de letra o dígito que representa la imagen.
5. Una vez procesadas todas las imágenes, se codifica el vector de clases "*y*" mediante la técnica *one-hot enconding*, de tal manera que el vector pasa a convertirse en una matriz en la que cada fila es a su vez un vector de 36 elementos donde todas las posiciones tienen valor 0 salvo la correspondiente a la clase de la letra o número de la imagen, que vale 1. Esta codificación se lleva a cabo mediante la función *LabelBinarizer* de la librería *Sci-kit Learn*.
6. Finalmente se exporta la matriz de las imágenes (*X*) y la matriz de clases (*y*) a dos ficheros de texto para facilitar su reutilización durante el proceso de entrenamiento de la red.

8.2 Entrenamiento de la red

El proceso de entrenamiento de la red CNN del sistema se basa en la combinación del algoritmo *backpropagation* con el método de descenso del gradiente ambos vistos en el apartado **3.4**.

Se trata básicamente de un problema de optimización en el que iterativamente se buscan los valores de los parámetros – pesos y *bias* de la red - que hacen que el valor de la función de error o coste sea mínimo. En el caso de la red CNN del sistema, la función elegida para medir lo buena o mala que es la clasificación de las imágenes es el error de **entropía cruzada** (*cross entropy error*), una de las habituales en el diseño de redes neuronales convolucionales.

De las distintas variantes del método de descenso del gradiente se ha elegido la versión **estocástica o incremental**, que consiste en calcular iterativamente el gradiente para un subconjunto o *batch* de los datos de entrenamiento, actualizando los pesos de la red mediante la propagación de errores hacia atrás, sin esperar a que se procese el conjunto de datos completo.

Desde el punto de vista del coste computacional, este método es más eficiente que procesar el gradiente en cada iteración el *dataset* completo, ya que suele ofrecer una convergencia más rápida. Aunque también es cierto que corre el riesgo de quedarse “atascado” en un mínimo local de la función de error.

8.2.1 Inicialización

Antes de ejecutar el proceso de entrenamiento es necesario generar e inicializa mediante *TensorFlow* la arquitectura de red presentada en el apartado **7.3**.

De manera resumida, en el módulo de código – página **46** - se declara un **grafo de operaciones** que contiene tanto las variables que almacenan los datos de entrenamiento, matrices de pesos, *bias* y salidas de la red; como las llamadas a las primitivas del *framework* para definir la función de error, las capas convolucionales (con su ventana, *stride*, filtros y funciones de activación), capas de *pooling*, capa FC, salida *softmax* y la llamada para actualizar los pesos mediante *backpropagation* y descenso del gradiente.

En la siguiente imagen se puede ver el grafo de la red CNN del sistema tal y como la representa la herramienta de visualización *TensorBoard* incluida en el *framework*:

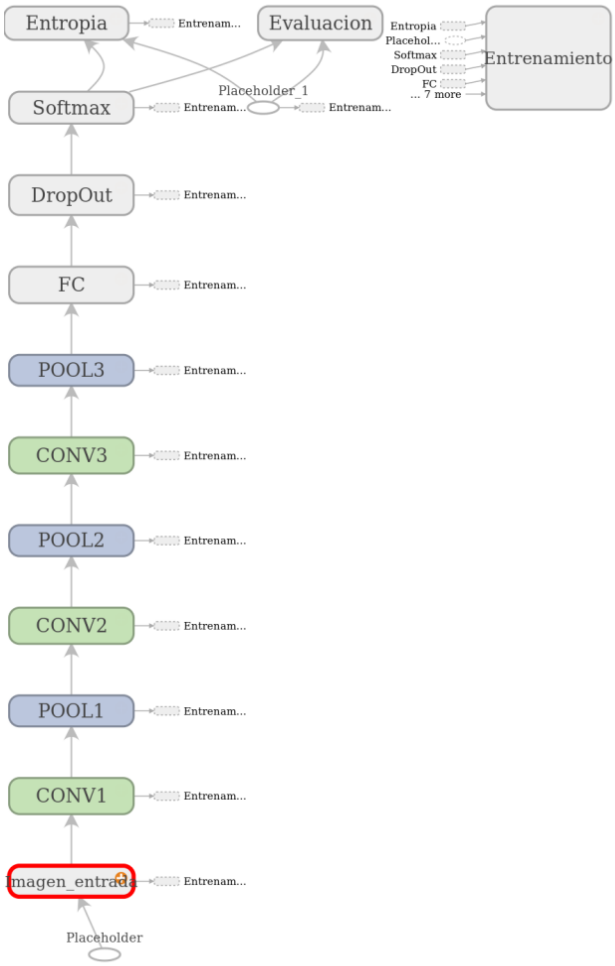

Figura 18 – Grafo computacional de TensorFlow

Aparte de declarar la estructura de la red, en el grafo de operaciones se indica además cómo debe ser inicializada. En el caso concreto de este sistema, se ha optado por asignar a los pesos pequeños valores aleatorios y a los *bias* valores ligeramente positivos.

El objetivo es evitar que durante el entrenamiento se produzca el conocido problema de las "*neuronas muertas*", en el que las neuronas calculan siempre el mismo valor de salida independientemente del valor de entrada, y suele estar originado por valores anómalos en los pesos de la red o por una tasa de aprendizaje inadecuada.

8.2.2 Entrenamiento

Una vez declara e inicializad la red, el siguiente paso es importar el conjunto de imágenes de entrenamiento pre-procesadas por el módulo de código del apartado **8.1.2**. Antes de iniciar el entrenamiento, este conjunto de imágenes se divide aleatoriamente en tres subconjuntos denominados *entrenamiento*, *validación* y *test*; el primero contiene un total de **28.012** imágenes, el segundo **3.113** y el tercero **3.459**.

El conjunto de entrenamiento se utiliza para determinar los pesos de la red que permitan clasificar con el menor error posible las imágenes, el de validación para comprobar si se produce sobreajuste (*overfitting*) durante el entrenamiento, y el conjunto de test para calcular la precisión del sistema a la hora de clasificar imágenes que no han sido analizadas durante las iteraciones de entrenamiento.

La sección de código encargada de ejecutar el proceso de entrenamiento – página **46** - realiza un total de **4.500** iteraciones seleccionando en cada una un lote o *batch* de **300** imágenes del conjunto de entrenamiento para optimizar los pesos de la red mediante la función *GradientDescentOptimizer* de *TensorFlow* con una tasa de aprendizaje de **0.0001**; lo que significa que al terminar el proceso cada imagen se habrá analizado unas **50 veces** aproximadamente.

Al igual que ocurre con la arquitectura de la red, tanto el número de iteraciones, como el tamaño de lote y la tasa de aprendizaje se han determinado de manera empírica comparando los resultados de distintas configuraciones - ver el apartado **0** para más información -.

8.3 Evaluación

Para evaluar la eficacia y eficiencia del proceso de aprendizaje de la red se ha optado por monitorizar la evolución de la precisión y el error general a la hora de clasificar las imágenes de los subconjuntos de *entrenamiento* y *validación* durante el proceso iterativo de entrenamiento.

El seguimiento de la precisión permite detectar si se está produciendo sobre-entrenamiento (*overfitting*) de la red, es decir, si los pesos y *bias* se están ajustando excesivamente a las particularidades de las imágenes del

subconjunto de entrenamiento, lo que puede provocar que una vez entrenada la red no sea capaz de clasificar correctamente imágenes distintas a las analizadas durante el aprendizaje.

En la siguiente imagen se puede ver la evolución de la **precisión** tanto en el conjunto de entrenamiento (línea azul) como en el de validación (línea verde) durante 4.500 iteraciones. Como se puede comprobar la distancia entre ambas es muy reducida, lo que a priori es un buen indicador de que no se ha producido *overfitting*.

También se puede observar la elevada precisión obtenida en ambos subconjuntos: aproximadamente 99% para el de entrenamiento y 98% para el de validación. Para el conjunto de imágenes de **test** la precisión obtenida con la configuración definida en el apartado anterior es de **97,83%**.

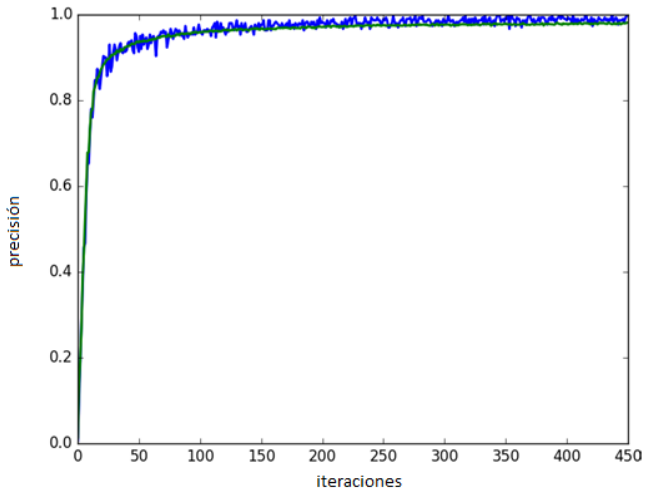

Figura 19 – Evolución de la precisión durante el entrenamiento

Respecto a la función de **error**, la siguiente gráfica muestra su evolución a lo largo de 4.500 iteraciones. En este caso se puede observar una caída muy pronunciada durante las primeras 500 iteraciones y bastante más suave pero constante durante el resto, lo que indica que en principio la tasa de aprendizaje elegida para el método del gradiente parece adecuada.

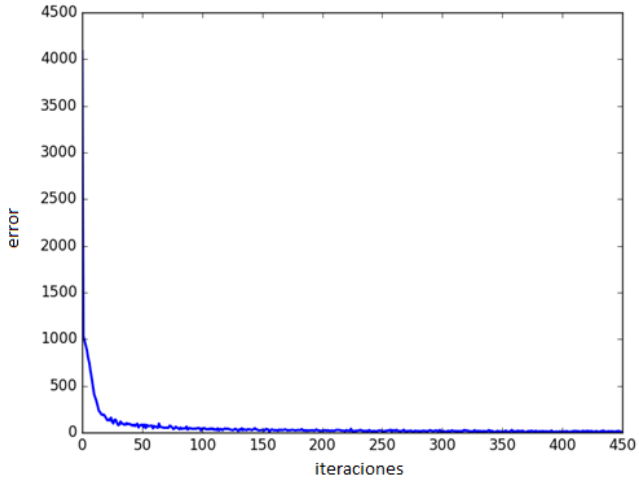

Figura 20 –Evolución de la función de error durante el entrenamiento

Además del seguimiento de las magnitudes anteriores, la evaluación de la eficacia del proceso de aprendizaje se completa ejecutando la red entrenada con una muestra de dígitos y letras extraídos de imágenes de matrículas sintéticas y reales (ver apartado **9** para más información).

8.4 Optimización

Como se comentaba en el apartado **8.2.2**, tanto las características de la arquitectura de red (capas, filtros, tamaño de las ventanas, etc.), como los parámetros del proceso de aprendizaje (número de iteraciones, tamaño de lote, tasa de aprendizaje, etc.), han sido determinados de manera empírica, en concreto siguiendo el procedimiento propuesto por Nikhil Buduma **[10]** que se muestra a continuación:

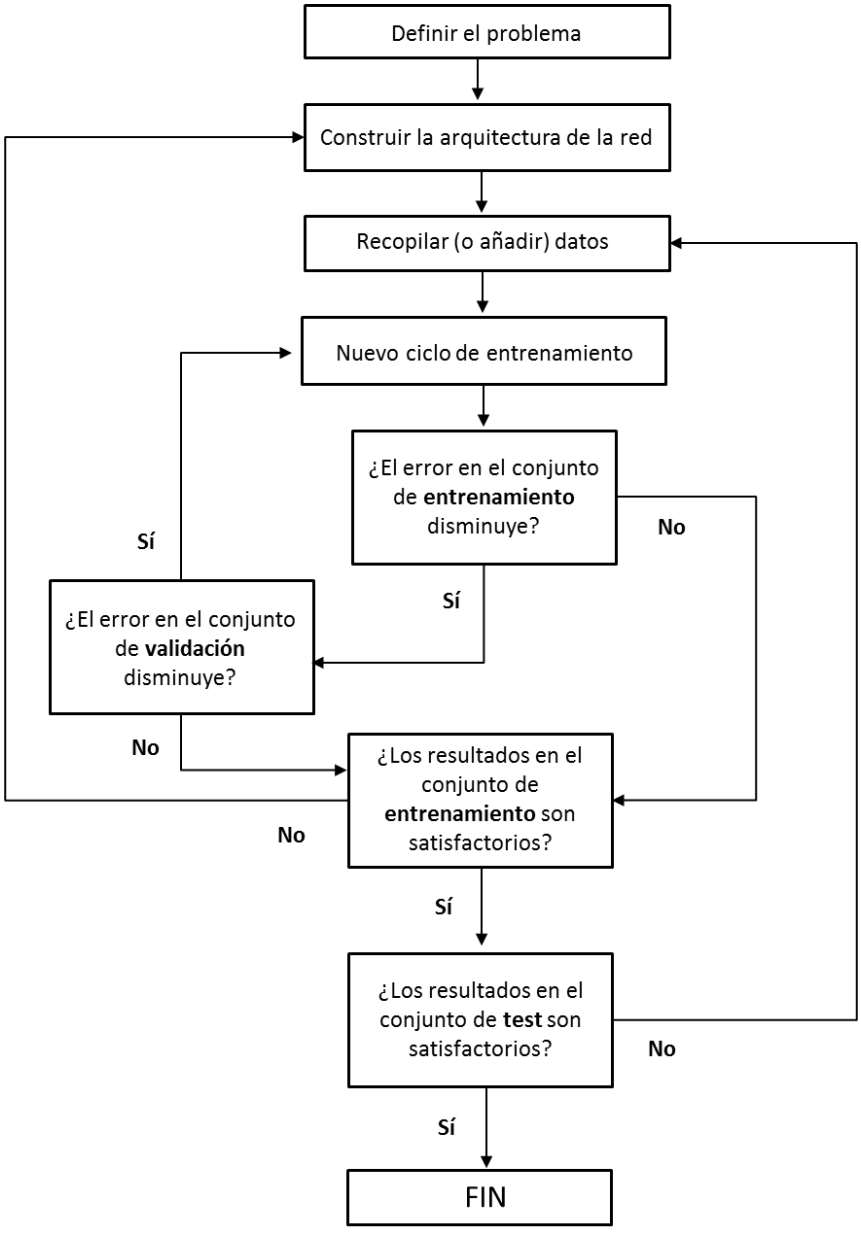

Figura 21 – Procedimiento seguido para diseñar y optimizar la red CNN
Fuente: "Fundamentals of Deep Learning". Nikhil Buduma. O'Reilly 2016

Como se puede ver en el diagrama anterior, se trata de ajustar iterativamente los parámetros de la arquitectura de la red y el proceso de entrenamiento en función de los resultados obtenidos. Para ello es necesario conocer lo eficaz que es el aprendizaje para cada ajuste, siendo fundamental contar con indicadores como los presentados en el apartado anterior.

Como ejemplo de este proceso, las siguientes tablas muestran la precisión obtenida en el conjunto de imágenes de test con distintos valores de tamaño de lote y número de iteraciones:

Iteraciones	Tamaño lote	Precisión %
4.500	300	97,83
4.500	200	97,72
4.500	100	96,55
4.500	50	94,54

Iteraciones	Tamaño lote	Precisión %
4.500	300	97,83
3.000	300	97,60
2.000	300	96,50
1.000	300	93,20

En la tabla de la izquierda se puede observar como el aumento del tamaño de lote para el mismo número de iteraciones supone una mejora en la precisión de la clasificación del subconjunto de test. Esto se debe a que se produce un aumento en el número de *epochs* o veces que se analiza cada imagen de entrenamiento durante el aprendizaje.

Respecto a la tabla de le derecha, es importante señalar que el número máximo de iteraciones elegido (4.500) no es casual, ya que durante el proceso de optimización se pudo observar como valores superiores a esta cifra aumentaban la probabilidad de que se produjera el efecto de las "neuronas muertas" comentado en el apartado **8.2.1**, tal y como refleja la siguiente gráfica de seguimiento de la precisión obtenida para 5.000 iteraciones:

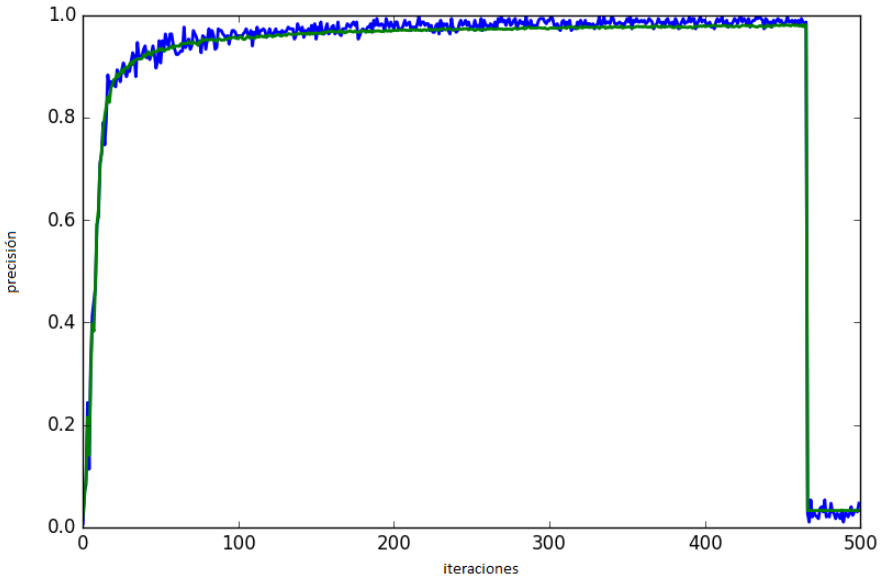

Figura 22 – Efecto de las "neuronas muertas" en la precisión de la red

Aparte del proceso de ajuste del diagrama anterior, para la optimización de la red también se han seguido algunas de las recomendaciones y técnicas habituales en el diseño de redes neuronales convolucionales, concretamente:

- Revisión del conjunto de datos de entrenamiento para eliminar imágenes no representativas del problema. En este caso son las representaciones exóticas o ambiguas de caracteres comentadas en el apartado **6.1**.
  Para detectarlas, aparte de revisar manualmente el *dataset*, ha sido de gran ayuda evaluar los errores cometidos por la red con muestras de imágenes de matrículas durante el entrenamiento, tal y como se indicaba en el apartado **8.3**.

- Centrado de los datos de los conjuntos de entrenamiento, test y validación. Se trata de una operación habitual en los algoritmos de aprendizaje de máquina que consiste en restar a las matrices de datos la media de cada columna. Este centrado suele venir acompañado por una normalización de datos, que en el caso de imágenes no es necesaria ya que los valores de los píxeles están en la misma escala (0-255).

- *Dropout* de las neuronas de la capa totalmente conectada (FC). Se trata de una técnica bastante reciente para reducir el sobre-entrenamiento. Fue presentada por Srivastava et al. **[11]** en 2014 y consiste en "apagar" aleatoriamente algunas neuronas durante el entrenamiento, de tal manera que sólo se actualicen los pesos de las que quedan activas.

9. Resultados obtenidos

Una vez entrenada la red CNN tan sólo queda comprobar la eficacia del sistema de reconocimiento de matrículas diseñado. Para ello se han utilizado tres colecciones de imágenes, dos de ellas creadas con las herramientas comentadas en el apartado **6.2**.

Los detalles de cada una son los siguientes:

1. Colección de 100 imágenes de matrículas generadas aleatoriamente por ordenador según el formato nacional vigente, es decir, dos grupos de caracteres constituidos por un número de cuatro cifras, desde el 0000 al 9999, y tres letras, en donde se suprimen las cinco vocales y las letras Ñ, Q, CH y LL.

2. Colección de 100 imágenes de matrículas generadas aleatoriamente por ordenador según distintos formatos estadounidenses. Dado que no existe una normativa única en este país, se ha optado por emplear uno de los formatos más comunes, que consiste en dos grupos de tres caracteres, el primero con números de 000 a 999, y el segundo con letras de AAA a ZZZ.

3. Muestra reducida de 10 imágenes de matrículas reales según el formato nacional vigente obtenidas con el consentimiento del propietario del vehículo.

En las tres colecciones de imágenes el nombre de los ficheros es el contenido de la matrícula seguido por la extensión del archivo (por ejemplo: *231HFG.png*).

El módulo de código encargado de comprobar la eficacia del sistema – página **42** - recorre uno a uno cada fichero, envía la imagen correspondiente a los módulos de segmentación y clasificación. Una vez recibida la respuesta del clasificador comprueba las diferencias entre el nombre del fichero (sin extensión) y la cadena con los caracteres reconocidos por el sistema.

Los resultados obtenidos para las tres colecciones son los siguientes:

Colección	Total imágenes	Aciertos	Fallos
Matriculas nacionales	100	100	0
Matriculas USA	100	60	40
Matriculas reales	10	10	0

Los resultados para las 100 imágenes en **formato nacional** son realmente espectaculares, obteniendo un **100%** de aciertos en el reconocimiento del contenido de estas imágenes de matrículas.

En el caso de las **imágenes reales** los resultados son igualmente excelentes - **100%** de aciertos -, aunque es importante señalar que esta muestra es demasiado pequeña para ser considerada significativa y además, las imágenes

han sido obtenidas en condiciones muy favorables. Llama la atención, sin embargo, como el sistema demuestra cierta tolerancia a leves rotaciones o deformaciones presentes en algunas imágenes, tal y como se puede comprobar en las siguientes imágenes de esta colección cuyo contenido fue reconocido correctamente:

Figura 23 – Imágenes tomadas con deformaciones leves que han sido reconocidas correctamente por el sistema

La evaluación del sistema con la colección de 100 matrículas en distintos **formatos estadounidenses** resulta bastante decepcionante, logrando tan solo un **60%** de aciertos. Sin embargo, un análisis detallado de los errores cometidos por el sistema deja ver como en la mayoría de casos se deben a un único carácter, y casi todos se pueden considerar "errores razonables", como confundir el número *2* y la letra *Z*, el *1* y la *I,* el *0* y la letra *O*, por ejemplo.

En la siguiente imagen se puede ver una de las matrículas reconocida incorrectamente. En este caso el sistema confundió la letra "*O*" con el número "*0*". Como se puede comprobar, la representación de ambos caracteres es prácticamente idéntica para este formato de matrícula:

Figura 24 – Matrícula reconocida incorrectamente al confundir el clasificador la letra "O" con el número "0"

En la siguiente tabla se puede ver la lista completa de errores cometidos en las matrículas de esta última colección:

Matrícula generada	Matrícula reconocida	Errores
532IUZ	53ZIUZ	1
779IBB	779I88	2
524QYR	5Z4QYR	1
173FMW	173FMV	1
290BKB	Z908K8	3
612HZX	61ZHZX	1
866DJJ	866OJJ	1
518ALP	5I8ALP	1
082COV	082C0V	1
292YDQ	Z9ZYO0	4
290CCJ	Z90CCJ	1
208CMT	Z08CMT	1
1610LX	1610LX	1

352NVR	35ZNVR	1
895DYX	895OYX	1
160CUD	160CUO	1
255ASV	Z55ASV	1
623ADG	6Z3AOG	2
370BXM	3708XM	1
973KOM	973K0M	1
582MBR	582M8R	1
199WWS	I99WWS	1
984VYW	984VYV	1
681XEB	681XE8	1
574UFO	574UF0	1
149LYK	I49LYK	1
905XUO	905XU0	1
627YAY	6Z7YAY	1
206HIA	Z06HIA	1
341UFD	341UFO	1
826AKO	826AK0	1
775RWG	775RYG	1
550WDD	550WOO	2
420OKQ	4200KQ	1
163QNZ	I63QNZ	1
557PBN	557P8N	1
814RWS	814RYS	1
026CZD	0Z6CZD	1
421NDY	42INOY	2
123TVF	1Z3TVF	1

10. Conclusiones

A continuación se exponen de forma resumida las principales conclusiones extraídas como resultado del estudio y desarrollo del sistema propuesto en este trabajo:

1. La primera conclusión a destacar es la confirmación de la viabilidad de un sistema de reconocimiento automático de matrículas (ALPR) basado en una red neuronal convolucional. El uso de este tipo de redes permite crear un sistema realmente capaz de "aprender" las características de los distintos dígitos y letras de las matrículas, sin necesidad de complejos mecanismos de extracción de atributos. Tan solo son necesarios un *dataset* y una arquitectura de red adecuados al problema.

2. La segunda es la constatación de la importancia que tiene disponer de un conjunto de imágenes de entrenamiento suficientemente amplio y representativo del problema para obtener resultados satisfactorios con una red neuronal convolucional. En el caso del sistema presentado en este trabajo, el *dataset* empleado no era específico para el problema planteado, y prueba de ello es la gran cantidad de imágenes de caracteres en minúscula o con fuentes exóticas que fue necesario eliminar para optimizar el proceso de aprendizaje. Sin duda, un conjunto de imágenes diseñado a medida de las necesidades del problema habría ofrecido mejores resultados que los que se han obtenido.

3. La tercera conclusión es lo laborioso que puede llegar a ser optimizar el proceso de aprendizaje de una red neuronal convolucional, y la escasa literatura que hay al respecto actualmente. Durante el desarrollo de este TFM ha sido necesario repetir en múltiples ocasiones el proceso propuesto en el apartado **8.4** hasta dar con una arquitectura y ajustes del proceso de entrenamiento que ofrecieran resultados razonablemente satisfactorios.

4. Como cuarta conclusión poner de manifiesto como las librerías *NumPy*, *OpenCV* y especialmente *TensorFlow*, facilitan enormemente el trabajo a la hora de diseñar sistemas de clasificación de imágenes basados en redes neuronales. Operaciones complejas como las que llevan a cabo los algoritmos de tratamiento de imágenes, o el cálculo diferencial necesario para el entrenamiento mediante *backpropagation* y descenso del gradiente; quedan reducidas a simples llamadas a funciones y primitivas, permitiendo centrar todo el esfuerzo en el diseño y optimización del sistema.

5. La última conclusión que quiero destacar son las múltiples posibilidades de mejora que tiene el sistema propuesto. Este trabajo ha requerido una importante labor de documentación y estudio de dos materias nuevas para

mí: las redes neuronales convolucionales y la visión artificial. Esto evidentemente se ha reflejado en el sistema desarrollado, y con toda seguridad habrá mejores alternativas de diseño y programación para los módulos del sistema y por supuesto para el conjunto de datos de entrenamiento elegido. Algunas de las mejoras y ampliaciones que se podrían plantear son:

- Añadir un módulo previo al de segmentación de caracteres que permita localizar automáticamente matrículas en fotografías de vehículos.
- Revisar el módulo de segmentación para evitar el uso de medidas absolutas en píxeles para detectar regiones de interés.
- Ampliar el *dataset* de entrenamiento, recopilando nuevas imágenes o mediante el uso de técnicas como "*data augmentation*", para crear nuevas imágenes a partir de transformaciones de las existentes, lo que sin duda mejoraría el porcentaje de aciertos del sistema.
- Probar otras configuraciones de red con distintos parámetros, número de capas, funciones de activación o algoritmo de aprendizaje.

Por último, señalar que realizar este TFM ha sido una experiencia muy positiva, ya que me ha permitido dar mis primeros pasos en este apasionante mundo del "*Deep Learning*", al que espero poder dedicar más tiempo en el futuro.

11. Bibliografía

[1] ImageNet Project. http://image-net.org - Marzo 2016.

[2] ImageNet ILSVRC2015. http://image-net.org/challenges/LSVRC/2015/results - Marzo 2016.

[3] Chars74k dataset. T. E. de Campos, B. R. Babu, M. Varma. http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/ - Marzo 2016.

[4] Informe 425/2006. Agencia Española de Protección de Datos (AGPD). https://www.agpd.es/portalwebAGPD/canaldocumentacion/informes_juridicos/conceptos/common/pdfs/2006-0425_Matr-ii-culas-de-veh-ii-culos-y-concepto-de-dato-de-car-aa-cter-personal.pdf - Abril 2016

[5] Platesmania License Plate Generator. http://platesmania.com/es/informer - Abril 2016

[6] Acme License Maker. http://acme.com/licensemaker - Abril 2016

[7] ImageNet Classification with Deep Convolutional Neural Networks. Krizhevsky et al. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf - Marzo 2016

[8] Gradient-Based Learning Applied to Document Recognition. LeCun et al. http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf - Marzo 2016

[9] Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. Simard et al. http://research.microsoft.com/pubs/68920/icdar03.pdf -Abril 2016

[10] Fundamentals of Deep Learning. Nikhil Buduma. O'Reilly. http://shop.oreilly.com/product/0636920039709.do - Abril 2016

[11] Dropout: A simple Way to Prevent Neural Networks from Overfitting. Srivastava et al. https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf - Mayo 2016

[12] CS231n: Convolutional Neural Networks for Visual Recognition of Standford University. Fei-Fei Li, Andrej Karpathy. http://cs231n.github.io/ - Marzo 2016

[13] Hello World en Tensorflow. Jordi Torres. Watch this space. 2016. http://www.jorditorres.org/libro-hello-world-en-tensorflow/ - Marzo 2016

[14] TensorFlow Tutorials. https://www.tensorflow.org/versions/r0.7/tutorials/index.html - Marzo 2016

[15] Starts on TensorBoard. Rob Romijnders. http://robromijnders.github.io/tensorflow_basic/ - Abril 2016

[16] OpenCV – Python Tutorials. Alexander Mordvintsev, Abid K. Revision. https://opencv-python-tutroals.readthedocs.io/en/latest/index.html - Abril 2016.

[17] Simple Digit Recognition OCR in OpenCV-Python. Abid Rahman. http://stackoverflow.com/questions/9413216/simple-digit-recognition-ocr-in-opencv-python - Abril 2016

[18] Deep Learning Book. Ian Goodfellow, Yoshua Bengio, Aaron Courville. MIT Press. http://www.deeplearningbook.org – Marzo 2016

[19] ¿Qué es y cómo funciona Deep Learning? Rubén López. https://rubenlopezg.wordpress.com/2014/05/07/que-es-y-como-funciona-deep-learning/ - Marzo 2016

[20] Redes neuronales y sistemas borrosos. Bonifacio Martín del Brío y Alfredo Sanz Molina. Editorial RA-MA. 3º Edición 2006.

[21] Python Machine Learning. Sebastian Raschka. Packt Publishing 2015

[22] Practical Machine Learning. Sunila Gollapudi. Packt Publishing 2016.

[23] Machine Learning in action. Peter Harrington. Manning Publications. 2012

[24] Learning OpenCV. Gary Bradski and Adrian Kaehler. O'Reilly 2008.

[25] Introducción a la visión artificial. Nicolás Luis Fernández García. http://www.uco.es/users/ma1fegan/2011-2012/vision/Temas/Vision-artificial.pdf - Marzo 2016

[26] Wikipedia.org. http://wikipedia.org – Marzo 2016

Anexo – código fuente del sistema

Módulo de entrada al sistema

modulo_entrada.py

```
# -*- coding: utf-8 -*-
"""
@author: Francisco José Núñez Sánchez-Agustino
"""

import os
import modulo_segmentacion as segmentacion
import modulo_CNN as CNN

def reconocer_matriculas (ruta_matriculas):

    total_aciertos = 0
    total_fallos = 0
    caracteres_distintos = 0

    for ruta, subdirs, ficheros in os.walk(ruta_matriculas):

        subdirs.sort()

        for nombre_fichero in ficheros:

            ruta_completa = os.path.join(ruta, nombre_fichero)
            contenido_matricula = nombre_fichero.rsplit('.', 1)[0]
            caracteres_matricula = segmentacion.cargar_contenido(ruta_completa)
            matricula_reconocida = CNN.reconocer_matricula(caracteres_matricula)

            if contenido_matricula == matricula_reconocida:
                print "\nCORRECTO: ",contenido_matricula, " = ", matricula_reconocida
                total_aciertos = total_aciertos + 1
            else:
                caracteres_distintos = \
                sum(1 for x,y in zip(contenido_matricula,matricula_reconocida) if x != y)
                print "\n* ERROR: ", contenido_matricula," <> ",matricula_reconocida
                print "* CARACTERES DISTINTOS: ", caracteres_distintos
                total_fallos = total_fallos + 1

    print "\n***************"
    print "***************"
    print " ACIERTOS:", total_aciertos
    print " FALLOS:", total_fallos


######## LLAMADA PRINCIPAL ############

reconocer_matriculas("matriculas/nacional/")
#reconocer_matriculas("matriculas/reales/")
#reconocer_matriculas("matriculas/usa/")
```

Módulo de segmentación de caracteres

modulo_segmentacion.py

```
# -*- coding: utf-8 -*-
"""
@author: Francisco José Núñez Sánchez-Agustino
"""

import cv2
import math
import numpy as np

def verificar_roi(r):

    # Relaciones de aspecto admitidas para los caracteres de la matrícula

    RELACION_ASPECTO = 0.6
    RELACION_ASPECTO_ERROR = 0.4
    RELACION_ASPECTO_MINIMA = 0.10
    RELACION_ASPECTO_MAXIMA = RELACION_ASPECTO + RELACION_ASPECTO * \
                              RELACION_ASPECTO_ERROR

    # Altura mínima y máxima de los caracteres de la matrícula

    ALTURA_MINIMA = 28
    ALTURA_MAXIMA = 42

    ANCHURA_MAXIMA = 35
    ANCHURA_MINIMA = 10

    # Relación de aspecto actual de la ROI detectada

    relacion_aspecto_roi = float(r.shape[1])/r.shape[0]

    #print ("Aspecto ROI:", relacion_aspecto_roi)
    #print("Altura:", r.shape[0])
    #print("Anchura:", r.shape[1])

    # Comprobamos si cumple las restricciones

    if  relacion_aspecto_roi >= RELACION_ASPECTO_MINIMA and \
        relacion_aspecto_roi < RELACION_ASPECTO_MAXIMA and \
        r.shape[0] >= ALTURA_MINIMA and r.shape[0] < ALTURA_MAXIMA and \
        r.shape[1] >= ANCHURA_MINIMA and r.shape[1] < ANCHURA_MAXIMA:
        return True
    else:
        return False


def centrar_roi(r):

    ALTURA_ROI = 28
    ANCHURA_ROI = 32
    ANCHO_BORDE = 4

    # Redimensionar ROI a 28 píxeles de altura manteniendo su aspecto

    relacion_aspecto_roi = float(r.shape[1]) / float(r.shape[0])
    nueva_altura = ALTURA_ROI
    nueva_anchura = int((nueva_altura * relacion_aspecto_roi) + 0.5)
    roi_redim = cv2.resize(r, (nueva_anchura,nueva_altura))
```

```
    # Binarizar ROI

    ret,roi_thresh = cv2.threshold(roi_redim,127,255,cv2.THRESH_BINARY_INV)

    # Calcular las dimensiones del borde necesario para centrar ROI en 32x32px

    b_top = ANCHO_BORDE
    b_bottom = ANCHO_BORDE
    b_left = int((ANCHURA_ROI - nueva_anchura)/2)
    b_right = int((ANCHURA_ROI - nueva_anchura)/2)

    # Aplicar borde al ROI

    roi_borde =
cv2.copyMakeBorder(roi_thresh,b_top,b_bottom,b_left,b_right,cv2.BORDER_CONSTANT,value=[0,0,0])

    # Redimensionar ROI a 32x32px

    roi_transformado = cv2.resize(roi_borde,(32,32))

    return roi_transformado


def cargar_contenido(ruta):

    # Área en píxeles que debe tener la imagen de la matrícula (235x50px)

    AREA_PIXELES = 11750.0
    ALTURA_MINIMA_CONTORNO = 28
    AREA_MINIMA_CONTORNO = 50

    imagen_original = cv2.imread(ruta)

    # Redimensionar imagen a resolución 242x54

    r_aspecto = float(imagen_original.shape[1]) / float(imagen_original.shape[0])
    nueva_altura = int(math.sqrt(AREA_PIXELES / r_aspecto) + 0.5)
    nueva_anchura = int((nueva_altura * r_aspecto) + 0.5)
    imagen_original = cv2.resize(imagen_original, (nueva_anchura,nueva_altura))

    # Preprocesamiento de la imagen para facilitar la detección de contornos

    imagen_grises = cv2.cvtColor(imagen_original,cv2.COLOR_BGR2GRAY)
    imagen_desenfocada = cv2.GaussianBlur(imagen_grises,(5,5),0)
    imagen_thresh = cv2.adaptiveThreshold(imagen_desenfocada,255,1,1,11,2)

    # Detección de contornos

    _,contornos,_ = cv2.findContours(imagen_thresh,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
    caracteres =  np.empty((0,1024))
    posicion_caracteres = []
    total_correcto = 0

    for cnt in contornos:

       if cv2.contourArea(cnt) > AREA_MINIMA_CONTORNO:

            [x,y,w,h] = cv2.boundingRect(cnt)

            if  h > ALTURA_MINIMA_CONTORNO:

                cv2.rectangle(imagen_original,(x,y),(x+w,y+h),(0,0,255),2)
                roi = imagen_desenfocada[y:y+h,x:x+w]

                if verificar_roi(roi):
```

```
                    # VISUALIZAR CONTORNOS PARA DEPURAR
                    cv2.imshow('Resultado',imagen_original)
                    cv2.waitKey(0)

                    total_correcto = total_correcto + 1

                    roi_transformado = centrar_roi(roi)
                    caracter = roi_transformado.reshape((1,1024))
                    caracteres = np.append(caracteres,caracter,0)

                    # Guardamos en el vector la coordenada "x" del caracter

                    posicion_caracteres.append(x)

    # Añadimos la columna con la coordenada "x" del caracter detectado

    caracteres = np.c_[caracteres,posicion_caracteres]

    # Ordenamos por la columna que guarda la posición

    caracteres = caracteres[caracteres[:,1024].argsort()]

    # Eliminamos la columna con la posición después de ordenar

    caracteres = np.delete(caracteres,np.s_[1024], axis = 1)

    return caracteres

######## LLAMADA PARA DEPURAR EL MODULO #########

#cargar_contenido("matriculas/nacional/0081ZJW.png")
```

Módulo del clasificador de imágenes (red neuronal convolucional)

modulo_CNN.py

```
# -*- coding: utf-8 -*-
"""
@author: Francisco José Núñez Sánchez-Agustino

Basado en las siguientes referencias:

https://www.tensorflow.org/versions/r0.7/tutorials/mnist/pros/index.html
http://robromijnders.github.io/tensorflow_basic/

"""

import carga_entrenamiento as dataset
import tensorflow as tf
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import numpy as np


def weight_variable(shape,name):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial,name=name)

def bias_variable(shape,name):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial,name=name)

def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1], padding='SAME')

# Resetear el grafo de computación
tf.reset_default_graph()

# Declarar sesión
sess = tf.Session()

# Placeholders para imágenes y clases de entrenamiento
x = tf.placeholder("float", shape=[None, 1024])
y_ = tf.placeholder("float", shape=[None, 36])


# Inicio de la declaracion de la arquitectura de red

with tf.name_scope("Reshaping_data") as scope:
    x_image = tf.reshape(x, [-1,32,32,1])


with tf.name_scope("Conv1") as scope:
    W_conv1 = weight_variable([5, 5, 1, 64],"Conv_Layer_1")
    b_conv1 = bias_variable([64],"Bias_Conv_Layer_1")
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
    h_pool1 = max_pool_2x2(h_conv1)

with tf.name_scope("Conv2") as scope:
    W_conv2 = weight_variable([3, 3, 64, 64],"Conv_Layer_2")
    b_conv2 = bias_variable([64],"Bias_Conv_Layer_2")
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
    h_pool2 = max_pool_2x2(h_conv2)
```

```
with tf.name_scope("Conv3") as scope:
    W_conv3 = weight_variable([3, 3, 64, 64],"Conv_Layer_3")
    b_conv3 = bias_variable([64],"Bias_Conv_Layer_3")
    h_conv3 = tf.nn.relu(conv2d(h_pool2, W_conv3) + b_conv3)
    h_pool3 = max_pool_2x2(h_conv3)


with tf.name_scope("Fully_Connected1") as scope:
    W_fc1 = weight_variable([4 * 4 * 64, 1024],"Fully_Connected_layer_1")
    b_fc1 = bias_variable([1024],"Bias_Fully_Connected1")
    h_pool3_flat = tf.reshape(h_pool3, [-1, 4*4*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool3_flat, W_fc1) + b_fc1)


with tf.name_scope("Fully_Connected2") as scope:
    keep_prob = tf.placeholder("float")
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
    W_fc2 = weight_variable([1024, 36],"Fully_Connected_layer_2")
    b_fc2 = bias_variable([36],"Bias_Fully_Connected2")


with tf.name_scope("Final_Softmax") as scope:
    y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)


with tf.name_scope("Entropy") as scope:
    cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))

with tf.name_scope("train") as scope:
    train_step = tf.train.GradientDescentOptimizer(1e-4).minimize(cross_entropy)


with tf.name_scope("evaluating") as scope:
    correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))


saver = tf.train.Saver()

# Inicialización del grafo

sess.run(tf.initialize_all_variables())

indice = 0
batch_size = 300
total_epoch = 4500

resultados_accuracy_training =  []
resultados_accuracy_validation =  []
resultados_loss_train = []

# Recuperar red previamente entrenada (si existe)

ckpt = tf.train.get_checkpoint_state("CNN_log/")

if ckpt and ckpt.model_checkpoint_path:

    saver.restore(sess, ckpt.model_checkpoint_path)

else:

    print "no se ha encontrado checkpoint"

    X_train, X_val, X_test, y_train, y_val, y_test = dataset.cargar_dataset()
```

```
    # Centrado de datos de los subconjuntos

    X_train = X_train - np.mean(X_train, axis=0)
    X_val = X_val - np.mean(X_val, axis=0)
    X_test = X_test - np.mean(X_test, axis=0)

    # Bucle de iteraciones de entrenamiento

    for i in range(total_epoch):

        # Carga del batch de imágenes
        batch_x = X_train[indice:indice + batch_size]
        batch_y = y_train[indice:indice + batch_size]

        # Actualizamos el índice

        indice = indice + batch_size + 1

        if indice > X_train.shape[0]:
            indice = 0
            X_train, y_train = shuffle(X_train, y_train, random_state=0)

        if i%10 == 0:

            results_train = sess.run([accuracy,cross_entropy],feed_dict={x:batch_x,
y_: batch_y, keep_prob: 1.0})
            train_validation = sess.run(accuracy,feed_dict={x:X_val, y_: y_val,
keep_prob: 1.0})

            train_accuracy = results_train[0]
            train_loss = results_train[1]

            resultados_accuracy_training.append(train_accuracy)
            resultados_accuracy_validation.append(train_validation)
            resultados_loss_train.append(train_loss)

            print("step %d, training accuracy %g"%(i, train_accuracy))
            print("step %d, validation accuracy %g"%(i, train_validation))
            print("step %d, loss %g"%(i, train_loss))

        # Guardar el modelo en cada iteración del entrenamiento
        saver.save(sess, 'CNN_log/model.ckpt', global_step=i+1)

        sess.run(train_step,feed_dict={x: batch_x, y_: batch_y, keep_prob: 0.5})

    print ("FINALIZADO training")

    # Visualizar precisión y error para subconjuntos de entrenamiento y validación

    eje_x = np.arange(total_epoch/10)
    array_training = np.asanyarray(resultados_accuracy_training)
    array_validation = np.asanyarray(resultados_accuracy_validation)
    array_loss_train = np.asanyarray(resultados_loss_train)

    plt.figure(1)
    linea_train, = plt.plot(eje_x,array_training[eje_x],label="train",linewidth=2)
    linea_test, = plt.plot(eje_x,array_validation[eje_x],label="validation",linewidth=2)
    plt.legend(bbox_to_anchor=(1, 1.02), loc='upper left', ncol=1)
    plt.xlabel('epochs')
    plt.ylabel('accuracy')
    plt.show()

    plt.figure(2)
    linea_loss, = plt.plot(eje_x,array_loss_train[eje_x],label="loss",linewidth=2)
    plt.legend(bbox_to_anchor=(1,1.02), loc='upper left', ncol=1)
    plt.xlabel('epochs')
```

```
    plt.ylabel('loss')
    plt.show()


    # Calcular precisión para el subconjunto de test

    test_accuracy = sess.run( accuracy, feed_dict={x:X_test, y_: y_test, keep_prob: 1.0})
    print("test accuracy %g"% test_accuracy)


def reconocer_matricula(letras_matricula):

    matricula = ""
    clases =
["0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F","G","H","I","J","K","L","M","N"
,"O","P","Q","R","S","T","U","V","W","X","Y","Z"]

    letras_matricula = np.matrix(letras_matricula)

    classification = sess.run(y_conv, feed_dict={x:letras_matricula,keep_prob:1.0})

    for p in range(classification.shape[0]):
        pred = sess.run(tf.argmax(classification[p,:], 0))
        matricula = matricula + clases[int(pred)]

    return matricula
```

Módulo para la carga del conjunto de imágenes de entrenamiento

carga_entrenamiento.py

```
# -*- coding: utf-8 -*-

import os
import numpy as np
from skimage import io
from skimage.transform import resize
from sklearn.cross_validation import train_test_split


def generar_dataset():

    X = []
    y = []

    imagen = []

    total = 0

    for ruta, subdirectorio, ficheros in os.walk('english/fnt/'):

        # Ordenar el contenido del subdirectorio

        subdirectorio.sort()

        # Se itera fichero a fichero por cada subdirectorio del dataset

        for nombreFichero in ficheros:

            # Extraemos el código de la clase del nombre del fichero
            clase = nombreFichero[3:nombreFichero.index('-')]
            y.append(float(clase))

            # Componer la ruta completa a la imagen
            rutaCompleta = os.path.join(ruta, nombreFichero)

            # Cargar la imagen y reducirla a 32x32 píxeles
            imagen = io.imread(rutaCompleta,flatten=True)
            imagen_reducida = resize(imagen,(32,32))

            # Invertir imagen
            imagen_reducida = 1 - imagen_reducida

            # Guardar imagen en la matriz como vector de 1024 píxeles
            X.append(imagen_reducida.reshape(1024,1))

            print (nombreFichero)
            total = total + 1

    print (total)

    # Convertir matriz de imágenes en array numpy
    X = np.array(X)
    X = X.reshape(X.shape[:2])

    print (X.shape)

    # Codificar el vector de clases como "one-hot encoding"
    from sklearn import preprocessing
    lb = preprocessing.LabelBinarizer()
    lb.fit(y)
    y = lb.transform(y)
```

```
    # Convertir vector de clases en matriz numpy
    y = np.array(y, dtype=float)

    # Guardar matrices como ficheros de texto
    np.savetxt('datos_x.txt', X)
    np.savetxt('datos_y.txt',y)


def cargar_dataset():

    # Comprobar si ya existen las matrices de datos

    if  not(os.path.isfile('datos_x.txt')) or \
        not(os.path.isfile('datos_y.txt')):
        generar_dataset()

    X = np.loadtxt('datos_x.txt')
    y = np.loadtxt('datos_y.txt')

    print (X.shape)

    # Generamos los conjuntos de datos de entrenamiento y test

    X_tr, X_test, y_tr, y_test = train_test_split(X, y, test_size=0.10,
random_state=42)

    # Dividimos el conjunto "Train" en subconjuntos de entrenamiento y validación

    X_train, X_val, y_train, y_val = train_test_split(X_tr, y_tr, test_size=0.10,
random_state=42)

    return X_train, X_val, X_test, y_train, y_val, y_test


######## LLAMADA PARA DEPURAR EL MODULO #########

#X_train, X_val, X_test, y_train, y_val, y_test = cargar_dataset()

# Dimensiones de X = (34584, 1024)
# Dimensiones de y = (34584, 36)
```