RECONOCIMIENTO DE SEÑALES DE TRÁFICO

Antonio González Hidalgo
Máster Universitario en Ingeniería computacional y matemática
Área de Inteligencia Artificial.

Antonio Burguera Burguera
Carles Ventura Royo
Oct. 2018

Universitat Oberta de Catalunya

UNIVERSITAT ROVIRA i VIRGILI

FICHA DEL TRABAJO FINAL DE MÁSTER

Título del trabajo:	Reconocimiento de señales de tráfico
Nombre del autor:	Antonio González Hidalgo
Nombre del consultor:	Antonio Burguera
Nombre del PRA:	Carles Ventura Royo
Fecha de presentación (mm/aaaa)	06/2019
Titulación o programa:	Máster Universitario en Ingeniería Computacional y Matemática
Área del trabajo final:	Inteligencia Artificial
Idioma del trabajo:	Español
Palabras clave:	Deep Learning, Keras, Redes neuronales, Clasificador, Detector, Señales Tráfico

Resumen del trabajo (máximo 250 palabras):

Este TFM presenta un sistema capaz de detectar y clasificar señales de tráfico a partir de imágenes estáticas.

El sistema propuesto se basa en dos redes neuronales: una de detección de señales y otra de clasificación. En una primera parte, el trabajo se centra en encontrar los parámetros que ofrecen una mejor precisión para cada una de las CNN.

Seguidamente se realiza una serie de ejemplos para comprobar el correcto funcionamiento de las CNN. Se analiza por separado cada CNN y finalmente, se implementa un ejemplo del funcionamiento global del sistema.

A Sandra Martínez, la persona que me ha hecho posible llegar a todas y cada una de las metas que me he propuesto. Gracias por tu comprensión, cariño y paciencia.

Tabla de contenido

Índice de Tablas

Índice de Ilustraciones

1. Introducción

Este primer capítulo introduce el trabajo, explicando su contexto y su justificación. Posteriormente, se aborda los objetivos propuestos y finalmente, se presenta un plan de trabajo de cómo alcanzar dichos objetivos.

1.1. Contexto y justificación del TFM

A mediados del siglo XIX la humanidad presenció la revolución industrial. Aquello cambió drásticamente la sociedad, modificando para siempre la forma de vida de millones de personas. Actualmente, la humanidad se encuentra a las puertas de una nueva revolución tecnológica: la conducción autónoma de vehículos. Sin duda el alcance de este hito supondrá cambios muy profundos en la sociedad actual. Es pronto para predecir cómo afectará a la manera que tenemos actualmente de desplazarnos o como alterará a la industria de los transportes. Sin duda, en los próximos años seremos testigos de una nueva revolución.

Sin embargo, otros pequeños avances se deben de alcanzar para poder hablar de una nueva revolución. Estos pequeños avances son: vehículos autónomos energéticamente, vehículos capaces de leer el entorno e interactuar con él, vehículos capaces de interpretar y ejecutar unas normas de circulación establecidas, etc. Entre todos ellos, este trabajo se centra en la siguiente: vehículos capaces de interpretar el entorno.

Exactamente, ¿qué se quiere decir con "Interpretar el entorno"? Esta pregunta tiene una respuesta bastante amplia ya que en un contexto de un vehículo 100% autónomo, la detección del entorno engloba la detección de la carretera, las líneas que definen los diferentes carriles, identificar otros vehículos y sus posiciones relativas, detectar desperfectos en la calzada, leer las diferentes señales de tráfico para respetar las normas del código de circulación vigente, etc.

Sobre este último punto de la interpretación del entorno se va a trabajar en este TFM. Sobre la necesidad de tener un sistema capaz de leer las señales de tráfico, interpretarlas y ejecutar las acciones oportunas para poder seguir circulando sin poner en peligro a ningún usuario de la vía pública.

1.2. Objetivos

El objetivo principal de este TFM es la obtención de un sistema capaz de reconocer señales de tráfico procedentes de una imagen estática. Para lograr el objetivo, se dividirá el problema en dos partes. La primera, un detector de señales de tráfico, detectará si en una fotografía existe o no una señal de tráfico. La segunda parte evaluará la foto para clasificar e identificar la señal de tráfico correspondiente. En la Ilustración 1 se adjunta un esquema del sistema diseñado.

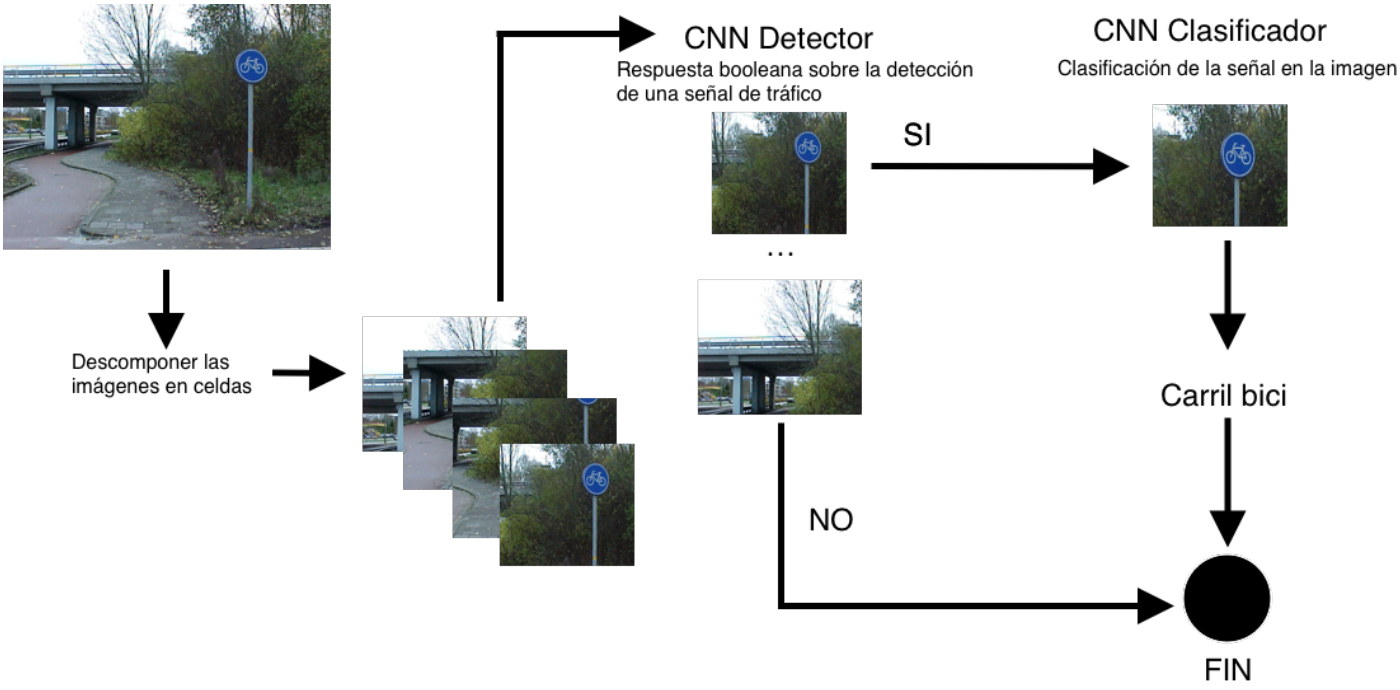


Ilustración 1. Esquema básico del sistema objetivo del TFM.

Una vez obtenido un sistema funcional, se estudiará la manera de ir mejorando la precisión del sistema sin aumentar drásticamente la cantidad de tiempo empleado en el proceso de detección. Finalmente, una parte importante del TFM será evaluar los resultados obtenidos, para ello, se realizarán una serie de pruebas y comparativas, que validarán, o no, las técnicas desarrolladas a lo largo de este trabajo.

1.3. Plan de trabajo

El plan de trabajo de este TFM se debe ceñir al calendario académico del curso 2018/19. Su realización debe comprender una duración anual con fecha de vencimiento en junio de 2019. Se deben de abordar los objetivos propuestos con un plazo alrededor de unos 7 meses. Hay que tener en cuenta que a lo largo de estos meses se deben de alcanzar ciertos hitos en unas determinadas fechas. Estos hitos se denominan PECs. Así el calendario queda definido por las fechas mostradas en la Tabla 1. Desglose de las PECs.

PEC #	Duración	Fecha de Inicio	Fecha de Fin
PEC 0	8 días	14/11/18	23/11/18
PEC 1	16 días	24/11/18	14/12/18
PEC 2	61 días	15/12/18	8/3/19
PEC 3	61 días	9/3/19	31/5/19
PEC 4	16 días	1/6/19	21/6/19
PEC 5	7 días	22/6/19	29/6/19
PEC 5b	12 días	30/6/19	14/7/19

Tabla 1. Desglose de las PECs

Para facilitar la organización del trabajo se han desglosado las tareas a lo largo de las PECs correspondientes. Usando el software de Microsoft Project se ha defino las tareas mostradas en la Ilustración 2.

Nombre de tarea	Duración	Comienzo	Fin
PAC 0	8 días	mié 14/11/18	vie 23/11/18
Redación del primer borrador	8 días	mié 14/11/18	vie 23/11/18
PAC1	16 días	sáb 24/11/18	vie 14/12/18
Curso apoyo de Deep Learning	14 días	lun 3/12/18	jue 20/12/18
Establer calendario de TFM	7 días	sáb 1/12/18	lun 10/12/18
Redación PAC1	5 días	lun 10/12/18	vie 14/12/18
PAC2	61 días	sáb 15/12/18	vie 8/3/19
Preparar & Redactar entorno	10 días	sáb 15/12/18	jue 27/12/18
Encontrar & Examinar datasets	5 días	vie 28/12/18	jue 3/1/19
Documentarse: Elaboración Red Neuronal	22 días	mié 2/1/19	jue 31/1/19
Desarrollo Red neuronal	11 días	vie 1/2/19	vie 15/2/19
Testeo & Mediciones Red Neuronal	10 días	sáb 16/2/19	jue 28/2/19
Redactar PAC2	7 días	vie 1/3/19	sáb 9/3/19
PAC3	61 días	sáb 9/3/19	vie 31/5/19
Documentarse: Uso de otro motor de reconocimiento	8 días	sáb 9/3/19	mar 19/3/19
Desarrollo Red neuronal #2	19 días	mié 20/3/19	lun 15/4/19
Testo & Mediciones Red Neuronal #2	8 días	mar 16/4/19	jue 25/4/19
Comparativa y conclusiones	11 días	mié 1/5/19	mié 15/5/19
Redactar PAC3	12 días	jue 16/5/19	vie 31/5/19
PAC4	16 días	sáb 1/6/19	vie 21/6/19
Redacción de la memoria	16 días	sáb 1/6/19	vie 21/6/19
PAC 5	7 días	sáb 22/6/19	sáb 29/6/19
Elaboración de la presentación	7 días	sáb 22/6/19	sáb 29/6/19
PAC 5b	12 días	dom 30/6/19	dom 14/7/19
Defensa pública	12 días	dom 30/6/19	dom 14/7/19

Ilustración 2. Desglose de hitos secundarios

También se ha generado un diagrama de Gantt para tener una vista rápida del alcance de todo el proyecto. Ilustración 3

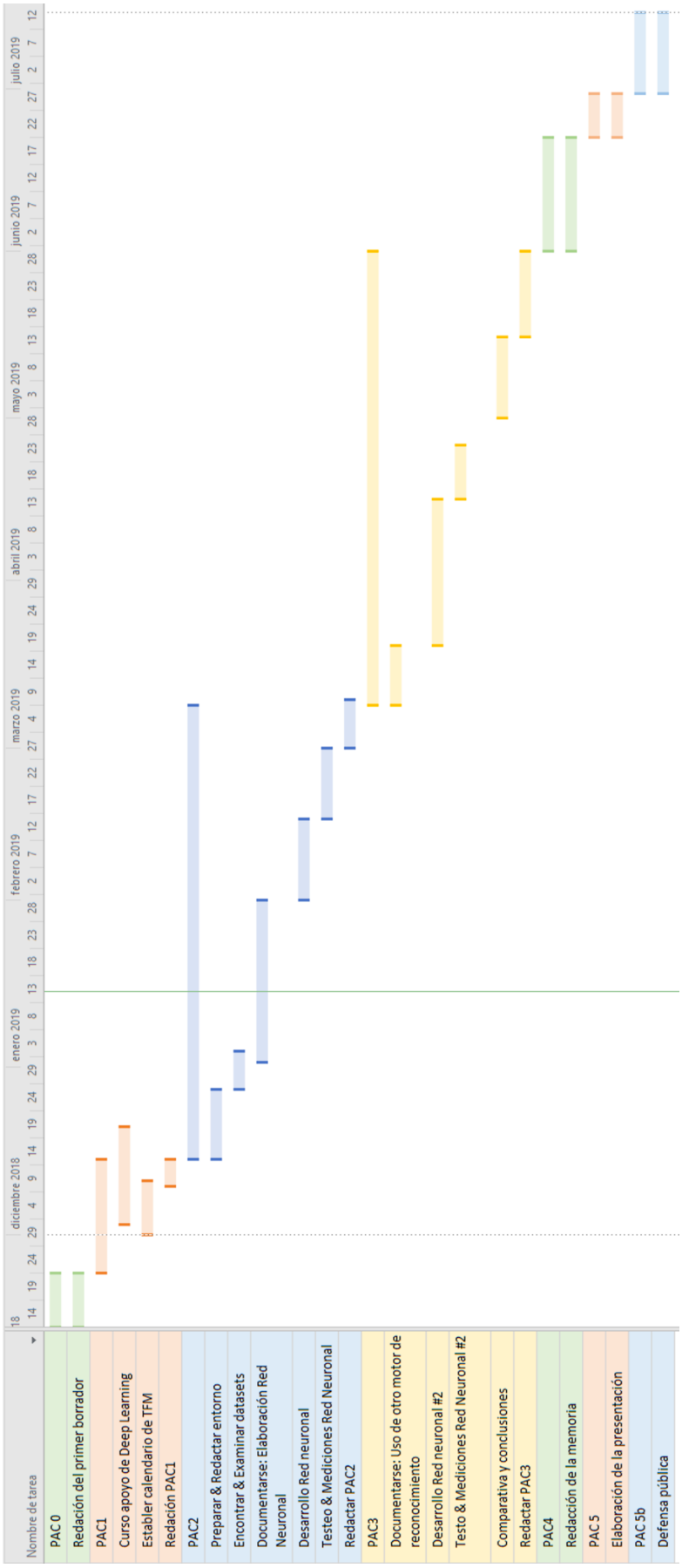


Ilustración 3. Diagrama de Gantt de la vida del proyecto.

1.4. Riesgos

Existen unos ciertos riesgos predecibles que se deben de tener en cuenta para evitar que estos afecten negativamente a la calendarización del proyecto. Es cierto, que quizás no se dan las características para la aparición de estos riesgos, pero se debe de tener un plan de contingencia para afrontarlos y absorberlos con la mayor naturalidad posible.

El riesgo principal es exceder el tiempo establecido para cada tarea. Se tendrá que detectar aquellos casos que, invirtiendo una gran cantidad de tiempo, se consiguen muy pocos resultados. El tema que abarca este trabajo es bastante amplio, por lo que sería muy sencillo caer en pozos de información redundante o aquellos que desvíen la atención del objetivo del proyecto. Se debe monitorizar el tiempo dedicado y la información recopilada de cada sección, para poder otorgar al lector una información suficiente y rigurosa sin caer en repeticiones.

Otro riesgo detectado, reside en aquellas tareas de investigación. Es fácil perderse en un mar de documentos, leer diversas webs, realizar multitud de pruebas y todo ello sin tener un rumbo o idea concisa de lo que se quiere investigar. A pesar de que estas tareas son de vital importancia, se deberá de respetar el tiempo establecido para las mismas.

El último riesgo por considerar reside en la propia documentación del trabajo. Priorizar implementación sobre documentación puede conducir a un desarrollo funcional que, sin embargo, es de escasa utilidad en el ámbito de la investigación y difícilmente ampliable en el futuro. Por ello es imprescindible que la documentación del proyecto se lleve a cabo a medida que alcanza la implementación.

La implementación del trabajo se centra en dos entornos de desarrollo: Docker [1] y VirtualEnv [2]. Se tratan con más detalle en el punto 11. Se priorizará la implementación con VirtualEnv debido a su fácil instalación y la rápida curva de aprendizaje. La implementación con Docker se producirá siempre y cuando no hayan surgido impedimentos que mermen el tiempo disponible.

Debido a que los entrenamientos de las redes neuronales consumen una gran cantidad de tiempo, se propone otra medida de contingencia: Reducir el volumen de los Datasets propuestos, eliminando de forma aleatoria un número proporcional de imágenes del conjunto de entrenamiento y prueba.

Durante el desarrollo se deberá prestar atención especial a los riesgos expuestos anteriormente. Intentando anticiparse ante cualquier imprevisto para disponer de suficiente margen de reacción.

2. Deep Learning

2.1. Introducción

Esta sección explica brevemente los conceptos básicos relacionados con el Deep Learning, así como el funcionamiento de una red neuronal sencilla.

2.2. Inteligencia artificial

Des de que se empezó a programar, el concepto de inteligencia artificial (IA) siempre ha estado presente entre investigadores e ingenieros. También ha sido foco de otros sectores, como el cine o la literatura, aunque en estos sectores se suele dar una visión mucho más futurista que la realidad de hoy en día.

Existen multitud de definiciones del término "Inteligencia artificial (IA)". Una aproximación más concisa y general es la que hace Jordi Torres en su libro Deep Learning – Introducción práctica con Keras [3]

> *La inteligencia artificial es el esfuerzo para automatizar tareas intelectuales normalmente realizadas por humanos*.

Esas tareas pueden ser de diferentes índoles y dependiendo de ello, el término IA se estructura en diferentes ramas. Este trabajo se centra específicamente en la rama Machine Learning.

2.3. Machine Learning

Machine Learning (ML) es la rama de la inteligencia artificial destinada al aprendizaje de las computadoras. Un algoritmo genérico de ML sería el estudio de una serie de datos para clasificar y predecir patrones. De esta manera sería posible clasificar nuevos datos. El algoritmo iría mejorando su precisión de manera paulatina en cada iteración. Dando la sensación de aprendizaje.

Según la Wikipedia [4] estos algoritmos de aprendizaje se pueden catalogar en los siguientes tipos: Aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semisupervisado y aprendizaje por refuerzo

Aprendizaje supervisado

Este tipo de algoritmos necesitan en su conjunto de entrenamiento, tener identificada la solución óptima. Algunos de los algoritmos más populares son: Las *Support Vector Machines (SVM)*, arboles de decisiones y las redes neuronales entre otros.

Aprendizaje no supervisado

En estos algoritmos los datos de entrenamiento no están etiquetados. De tal manera que el sistema no puede verificar la predicción con un resultado conocido previamente. Así que es

tarea del propio algoritmo obtener un resultado válido aplicando técnicas de clustering. Un ejemplo puede ser el algoritmo K-Means.

Aprendizaje semisupervisado

Se trata de una combinación de los dos algoritmos anteriores para obtener un resultado válido. En este tipo de algoritmos el conjunto de entrenamiento está divido en un conjunto de datos perfectamente etiquetados, como el aprendizaje supervisado. Otra parte de los datos no se encuentran etiquetados, replicando al aprendizaje no supervisado.

En algunos entornos se ha demostrado que contar con una muestra de datos etiquetados junto con otros no etiquetados mejora la precisión del sistema resultante.

Aprendizaje por refuerzo

Este algoritmo introduce la figura de un agente, que tendrá como objetivo indicar si la predicción realizada es válida o no. En caso de ser válida el aprendizaje por refuerzo se asemeja al aprendizaje supervisado. Sin embargo, en caso de ser no válida, el agente no indicará el valor correcto, pero sí que indicará la cuantía del error cometido. De este modo se obtiene un sistema que en cada iteración irá recompensando las decisiones correctas (aquellas que reducen el error) y penalizando las incorrectas (aquellas que aumentan el error).

2.4. Deep Learning

El Deep Learning es un tipo de IA, derivado del Machine Learning y es el concepto central de este trabajo. Consiste en elaborar un modelo, red neuronal, que en base a unas entradas (*features*) se obtendrá unas salidas (*labels*).

Para definir el Deep Learning, usaremos, nuevamente, la definición dada por Jordi Torres en su libro Deep Learning – Introducción práctica con Keras [3]:

> *Se trata de una estructura algorítmicas que permiten la recreación de modelos que están compuestos por múltiples capas de procesamiento para aprender representaciones de datos, con múltiples niveles de abstracción que realizan una serie de transformaciones lineales y no lineales a partir de los datos de entrada.*

El valor de la salida de una red neuronal será el resultado de unos cálculos iniciados en la primera capa, estos irán atravesando las capas internas de la red, hasta llegar a la capa de salida. Cada capa realizará una serie de transformaciones y cederá su salida a la siguiente, hasta recorrer todas las capas. Para el usuario, tan solo son visibles las capas de entrada y la capa de salida. El resto se denominan capas ocultas porque son invisibles al usuario. Se adjunta un ejemplo conceptual de red neuronal en la Ilustración 4.

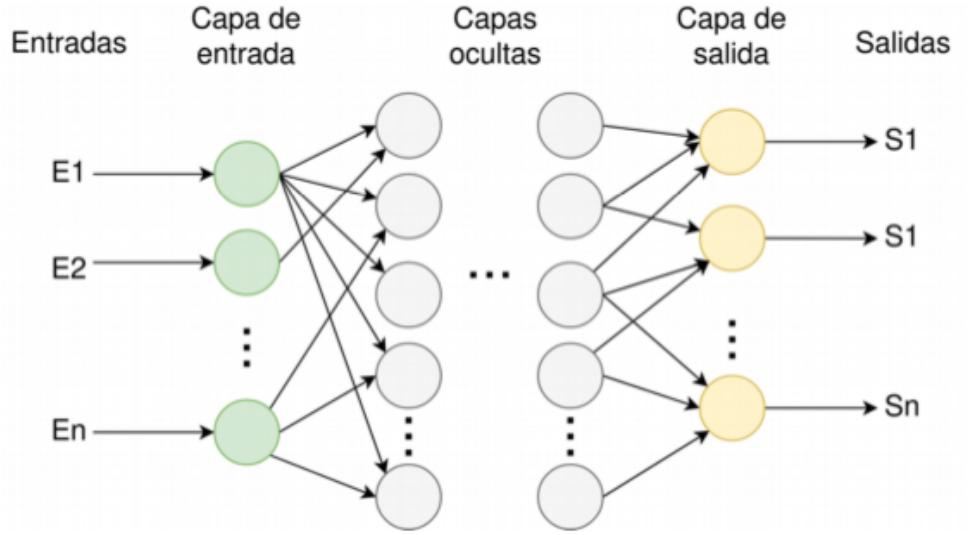


Ilustración 4. Ejemplo conceptual de red neuronal. Extraído de Redes Neuronales aplicadas al criptoanálisis del Advanced Encryption Standard por Pedro Novas Otero. [5]

Si tomamos un ejemplo sencillo, como el modelo de una progresión lineal ***y = wx + n***. Podemos ver cómo ***y*** toma el rol de salida (*label*). ***X*** es la variable de entrada (*feature*). ***W*** es uno de los parámetros que tiene que calcular la red neuronal (*weight*) y ***n*** toma el rol de sesgo (*bias*) que se trata de otro parámetro que debe de calcular la red neuronal. De esta manera, cada neurona de nuestra red tendrá una salida como la que se muestra en la Ecuación 1.

$$y = \sum_i w_i x_i + n$$

Ecuación 1. Expresión matemática de la salida de una neurona.

Sin embargo, en este punto se puede llegar a la conclusión equivocada de que la misma entrada siempre ofrecerá la misma salida. Para evitar esto las redes neuronales cuentan precisamente con una función de activación, cuyo objetivo es agregar la no linealidad al sistema determinando que funciones se deben de activar.

En la actualidad y gracias al Deep Learning se han conseguidos aplicaciones de gran utilidad que todo el mundo usa en mayor o menor medida. Por ejemplo: Google translate con su traducción simultánea, YouTube con su transcripción automática de los videos, Gmail a la hora de establecer los criterios de correo no deseado, Amazon en sus recomendaciones de productos o Netflix sobre sus recomendaciones de películas y series.

3. Herramientas

3.1. Introducción

Con el fin de lograr los objetivos propuestos surgen diversas preguntas: ¿Qué herramientas se van a usar? ¿Qué lenguajes de programación? o ¿Qué librerías son necesarias? Este apartado da respuesta a todas estas preguntas. Explica y justifica cada una de las elecciones tomadas. En el caso que el lector quiera obtener más detalle sobre la implementación realizada, se le remite al anexo 1 ubicado en el punto 11.

3.2. Python

Python [5] fue creado a finales de los años 80 por el científico de computación Guido van Rossum. Se trata de un lenguaje multiparadigma, con una sintaxis bastante sencilla. Tiene soporte multiplataforma y al igual que Java es un lenguaje interpretado.

Python además está respaldado por una gran comunidad de desarrolladores y científicos, quienes elaboran nuevas bibliotecas y complementos. Podemos encontrar una gran cantidad de bibliotecas de cálculo científico muy completas: NumPy [6], SciPy [7], Matplotlib [8], entre otras.

En el momento de escribir estas líneas, febrero de 2019, Python disfruta de gran popularidad dentro del mundo del desarrollo, encabezando rankings como el PYPL [9]. Donde se mide la popularidad de los diferentes lenguajes de programación en base a las búsquedas realizadas en Google.

Centrando el foco más en este trabajo. En Python se encuentra disponible la librería Keras [10], una de las librerías más extendidas a la hora de modelar redes neuronales. Ofrece un soporte sencillo y con una curva de aprendizaje bastante asequible. Keras será la librería principal de este trabajo.

Como IDE de programación se usará Jupyter [11], ya que otorga una interfaz muy cómoda basada en una página web. Desde ella se puede escribir, ejecutar y evaluar los resultados de una manera bastante sencilla.

3.3. Keras

Keras se trata de una API (*application programming interface*), escrita en Python, para la creación de redes neuronales de alto nivel. Requiere de un motor backend para la realización de los cálculos matemáticos. Los tres backends soportados son: Tensorflow [12], Theano [13] y CNTK [14]. Este trabajo usará el backend predeterminado de Keras: Tensorflow.

Keras no requiere de grandes centros de datos para su ejecución y puede ejecutarse en ordenadores personales, eso facilita el acceso a cualquier persona. Además, tiene soporte para utilizar la actual potencia de las GPU, mediante la tecnología CUDA [15].

Otro motivo por el que se debe de tener en cuenta Keras, es por su sencillez de uso y su curva de aprendizaje bastante suave. Además, en los últimos tiempos está ganando gran popularidad convirtiéndose en un estándar. Permite la creación de diferentes modelos de redes neuronales sin demasiadas conjeturas ni complicaciones. Este trabajo tratará dos tipos de redes: redes densamente conectadas y redes convolucionales.

Tensorflow

Es el backend predeterminado de la librería Keras. Se trata de una librería de código abierto desarrollada por Google. Es un sistema de aprendizaje de segunda generación y ofrece soporte para correr en múltiples CPUs y GPU.

Theano

Theano se trata de una librería escrita en Python por el '*Montreal Institute for Learning Algorithms'* (MILA). Su punto fuerte se basa en manipular y evaluar complejas expresiones matemáticas, haciendo mención especial al cálculo matricial. Theano también tiene soporte para correr en múltiples CPUs y GPU.

CNTK

The Microsoft Cognitive Toolkit (CNTK) se trata de un backend de Deep Learning creado por Microsoft Research. Actualmente tiene versión para Windows y para plataformas Linux. Al igual que Tensorflow y Theano, CNTK tiene soporte para correr en múltiples CPUs y GPU.

3.4. Datasets

Además de las herramientas comentadas, para crear la red neuronal, se debe de contar con algún conjunto de datos. La totalidad de estos datos se denomina población. Para el caso de estudio de este trabajo, los conjuntos de datos estarán constituidos por imágenes. Algunas de estas imágenes contendrán señales de trafico en diferentes posiciones, distancias y ángulos. Otras, mostrarán objetos que pueden encontrarse en la vía pública, cerca del entorno de las señales. Por ejemplo: coches, carreteras, arboles, etc.

Para entrenar cualquier red neuronal, la población se debe dividir en dos grandes conjuntos: El conjunto de entrenamiento y el conjunto de prueba. El primero tiene como objetivo entrenar la red neuronal dándole un conjunto de datos correctamente etiquetados, de esta manera la red "aprenderá" como debe de clasificar los datos recibidos. Una vez entrenada, la red neuronal usará los datos del conjunto de prueba para verificar que las clasificaciones realizadas se han realizado con éxito. Usando los aciertos y errores del conjunto de prueba se calcula la precisión de la red neuronal.

En esta tesis, se usarán dos conjuntos de datos: uno procedente del '*German Traffic Sign Detection Benchmark'* [16] y el otro es el '*Belgium TS Dataset*' [17]

The German Traffic Sign Detection Benchmark

Se trata de un conjunto de imágenes procedentes del "*Institut für Neuroinformatk*". Consta de un total de 51.839 imágenes divididas en: 39.209 para el conjunto de entrenamiento y 12.630 para el conjunto de prueba. Las imágenes están clasificadas en un total de 43 señales de tráfico. En la Ilustración 5 se muestra un ejemplo de cada una de las categorías de las señales.

Ilustración 5. Categorías de las señales de tráfico (dataset: The German Traffic Sign Detection Benchmark)

De manera adicional se ha generado un fichero .csv para identificar cada una de las señales mediante una breve descripción. El dataset ya preparado para este trabajo se encuentra en el repositorio de GitHub [18]

BelgiumTS Dataset

Se trata de un conjunto de datos mantenido por Radu Timofte. Consta de 4.575 imágenes para el conjunto de entrenamiento y 2.520 imágenes para el conjunto de pruebas, sumando un total de 7.095 imágenes. La Ilustración 6 muestra una señal correspondiente a cada una de las categorías.

Ilustración 6. Categorías de las señales de tráfico (dataset: BelgiumTS dataset)

También se ha generado un fichero .csv con el nombre de cada una de las categorías. Al igual que el anterior dataset, este también se encuentra disponible en el repositorio de GitHub [19]

Otros

Para el entrenamiento y test de la CNN detectora de señales de tráfico, es necesario disponer de un conjunto de imágenes donde se visualicen señales de tráfico y otro donde hubiera fotos del entorno sin mostrar señales de manera completa o parcial. Para suplir la primera categoría, podemos usar un extracto de los dos Datasets vistos anteriormente.

Para la segunda categoría se ha elaborado un conjunto propio. Partiendo del fichero TestIJCNN2013.zip del dataset '*German Traffic Sign Detection Benchmark GTSDB*' [20] que contiene 300 imágenes completas. Se adjunta un ejemplo en la Ilustración 7.

Ilustración 7. Imagen 00000.ppm del fichero TestIJCNN2013.zip

Usando Python, se ha programado un script para recorrer las 300 imágenes y recortar de manera aleatoria ventanas de una dimensión de 64x64 píxeles. En la sección 12.13 se encuentra el código del script. Algunos de estos recortes se adjuntan en la Ilustración 8.

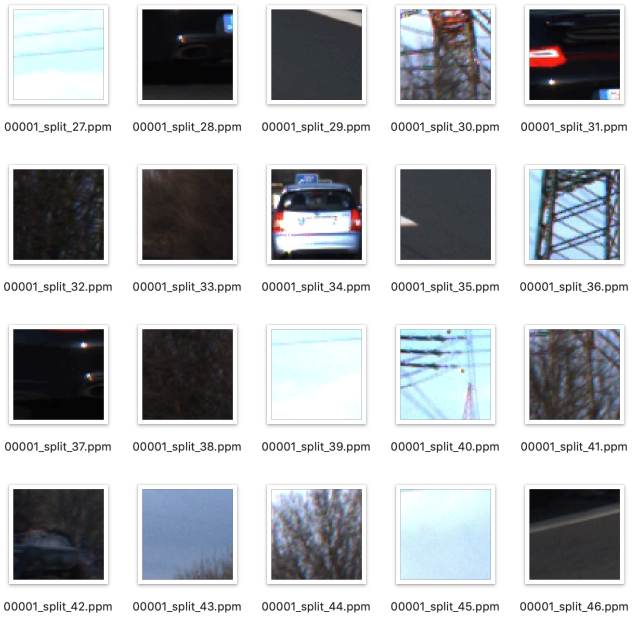


Ilustración 8. Ejemplo de recortes aleatorios generados

Una vez generado los recortes, se han evaluado manualmente para eliminar aquellos que coincidían o contenían parcialmente una señal de tráfico. Dejando finalmente 13.491 recortes válidos. Se han dividido en dos conjuntos: Entrenamiento (11.395 imágenes) y Prueba (2.096 imágenes).

Se han usado 4.086 imágenes con señales de tráfico, procedentes del '*Belgium TS Dataset*' [17], para completar el conjunto de entrenamiento y 12.631 imágenes procedentes del '*German Traffic Sign Detection Benchmark'* [16] para completar el conjunto de pruebas.

Combinando los recortes sin señales de tráfico generados y las imágenes extraídas de los dataset, se ha generado un tercer conjunto de datos para entrenar la red neuronal de detección. Las dimensiones finales del dataset son:

- Dataset para la CNN detectora: 30.208 imágenes.
    - Conjunto de entrenamiento: 15.481 imágenes.
        - Sin señales de tráfico: 11.395 imágenes.
        - Con señales de tráfico: 4.086 imágenes.
    - Conjunto de pruebas: 14.727 imágenes.
        - Sin señales de tráfico: 2.096 imágenes.
        - Con señales de tráfico: 12.631 imágenes.

El conjunto de datos ya preparando, así como el script de Python para generar las ventanas aleatorias de 64x64, se encuentra en el repositorio de GitHub [21].

4. Ajustes de hiperparámetros

4.1. Introducción

Este apartado explica las decisiones más relevantes a la hora de elaborar las redes neuronales. Por ejemplo, la elección de la función de activación, el optimizador, así como una serie de parámetros de aspecto más general pero esenciales para cualquier red neuronal.

La búsqueda de los hiperparámetros de una CNN es una búsqueda exhaustiva que requiere invertir una gran cantidad de tiempo computacional. Los hiperparámetros dependen unos de otros y para obtener los mejores resultados habría que realizar todas las combinaciones posibles entre todos ellos y para cada CNN que componen el sistema.

Con el fin de reducir el número de combinaciones a estudiar, se ha partido de una configuración base. Con ayuda de esta configuración se realiza un estudio para fijar la función de activación, el optimizador, el número de *epoch* y el *batch_size* de los sistemas de detección y clasificación.

La configuración inicial parte de un optimizador: *sgd* y unos valores para el *epoch* y el *batch_size* de 15 y 32 respectivamente.

4.2. Funciones de activación

En una red neuronal la función de activación tiene como objetivo introducir la no linealidad en las capacidades del modelo, determinando como se activan las neuronas de la red.

Keras permite configurar diferentes funciones de activación [22]. Utilizando la CNN de clasificación y de detección, se ha generado una prueba para comprobar que función de activación se ajusta mejor a los dos problemas planteados. Con los valores establecidos por defecto se ha realizado una batería de entrenamientos con todas las funciones de activación soportadas por Keras.

La perdida y precisión de los modelos entrenados se muestran en la Tabla 2. Se puede ver como para la CNN de clasificación la función de activación con mejores valores de precisión es la función '*selu'*. Así mismo, la CNN de detección obtiene unos mejores resultados con la función de activación '*elu'*.

Función Activación	CNN Clasificación		CNN Detector	
	BelgiumTS		Propio	
	Pérdida	Precisión	Pérdida	Precisión
sigmoid	3,691220478	0,024206349	1,044328167	0,142333288
linear	0,219508358	0,952777778	0,592144368	0,72035855
tanh	0,436195955	0,902380952	0,960378323	0,691837566
relu	0,286834762	0,938492063	0,937117609	0,654149124
selu	0,165745301	0,955952381	0,592177939	0,754855358
elu	0,196477469	0,953174603	0,544063501	0,768436778

softplus	0,867631597	0,803571429	13,8239539	0,142333288
softsign	1,128826431	0,70515873	0,876321035	0,699511069
hard_sigmoid	3,692276062	0,024206349	0,909453995	0,142333288

Tabla 2. Relación entre la función de activación y la precisión obtenida.

4.3. Optimizadores

A la hora de compilar un modelo de red neuronal es necesario pasar como mínimo dos parámetros: una función de pérdida o *loss* y un optimizador [23].

El proceso de entrenamiento de una red guarda muchas similitudes con el proceso de optimización de una función: ajustar los valores de entrada con pequeñas modificaciones e ir monitorizando la salida para encontrar un máximo o mínimo global de la función.

Keras tiene soporte para los siguientes optimizadores: SGD (Stochastic gradient descent), RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam. Utilizando la configuración por defecto más la función de activación fijada en el apartado 4.2, se ha entrenado diferentes modelos para buscar el optimizador que ofrece mejor rendimiento. En la Tabla 3 se adjunta los resultados obtenidos.

Optimizadores	CNN Clasificación		CNN Detector	
	BelgiumTS		Propio	
	Pérdida	Precisión	Pérdida	Precisión
sgd	0,139288217	0,965079365	0,66808268	0,73910091
rmsprop	0,167289924	0,981349206	1,81746826	0,84299878
adagrad	15,96458972	0,00952381	0,59958484	0,78249355
adadelta	0,158880846	0,978571429	1,20000043	0,80205079
adam	0,258498851	0,964285714	0,79568334	0,90499796
adamax	0,128896392	0,972619048	0,92338782	0,76694282
nadam	0,993733413	0,936111111	3,98619753	0,62861605

Tabla 3. Relación entre el optimizador y la precisión obtenida.

Se puede apreciar como el optimizador que ofrece mejor precisión para la CNN de clasificación es '*rmsprop*' mientras que para la CNN de detección el mejor optimizador '*adam*'.

4.4. Otros parámetros

Además de las funciones de activación y los optimizadores. Cualquier red neuronal necesitará otros parámetros básicos como:

- Epoch: indica cuantas veces van a pasar todos los datos del conjunto de entrenamiento por la red neuronal. Normalmente se suele ir incrementando el valor hasta que se produce un *overfitting* o sobreajuste.

- Batch_size: indica el tamaño de las particiones que se realizarán a los datos de entrenamiento y cada cuanto se actualiza el gradiente del modelo.

Utilizando los valores de función de activación y optimizador fijados en los anteriores apartados, se ha realizado una batería de pruebas para diagnosticar que valores de epoch y batch_size ofrecen las mejores precisiones. Para la CNN de clasificación, el conjunto de datos empleado es *BelgiumTS*. Por su parte, la CNN de detección ha usado el único conjunto de datos disponible para esta red. Los resultados de los modelos obtenidos se encuentran en la Tabla 4.

Epoch	Batch_size	CNN Clasificación		CNN Detector	
		BelgiumTS		Propio	
		Pérdida	Precisión	Pérdida	Precisión
20	32	0,16892847	0,98015873	7,22441914	0,51969306
25	32	0,25909917	0,975	1,92961037	0,86778487
30	32	0,24310301	0,9765873	3,74507655	0,75689257
20	64	0,1429517	0,9797619	1,01308006	0,85929648
25	64	0,13244304	0,98055556	1,13168625	0,88965096
30	64	0,17236458	0,975	1,79847682	0,85386391
20	128	0,10871683	0,97777778	0,73529777	0,7973652
25	128	0,13030479	0,98373016	0,76199001	0,8460546
30	128	0,19059645	0,975	1,48750028	0,84401738

Tabla 4. Relación entre el epoch, batch_size y la precisión.

Examinando los resultados podemos ver como para la CNN de clasificación la mejor configuración es usar un valor de 25 para el *epoch* y *batch_size* de 128. Sin embargo, la CNN de detección vuelve a tener mejores resultandos empleando otros valores diferentes a la CNN de clasificación. En este caso, un *epoch* igual a 25 y un *batch_size* de 64.

4.5. Red neuronal convolucional

Las redes convolucionales (CNN) se tratan de redes neuronales enfocadas al tratamiento de imágenes. Trabajan reconociendo pequeños patrones dentro de una imagen para intentar identificar un patrón más complejo. Por ejemplo, en el caso de querer identificar una cara, primero se deberá identificar un ojo, una nariz, una boca y saber las distancias y tamaños relativos unos respecto a otros.

Una CNN aparte de su capa de entrada y de salida, es necesario que tenga, al menos, dos capas ocultas que son básicas para este tipo de redes: una capa convolucional y una capa de pooling.

Capa convolucional

Esta capa tiene como objetivo detectar características de la imagen en pequeñas ventanas. Por ejemplo, aristas, líneas, etc. De esta manera, la unión de diversas capas convolucionales puede ir detectando dichas características hasta identificar un patrón más complejo.

El objetivo de la capa convolucional es crear un mapa de características de la imagen principal. En la Ilustración 9 se muestra un ejemplo del funcionamiento. El proceso de convolución recorre la capa de partida examinando pequeñas muestras, en el ejemplo se utiliza una matriz de 3x3 (representada en color rojo) que recorrerá de izquierda a derecha y de arriba a bajo la capa de partida.

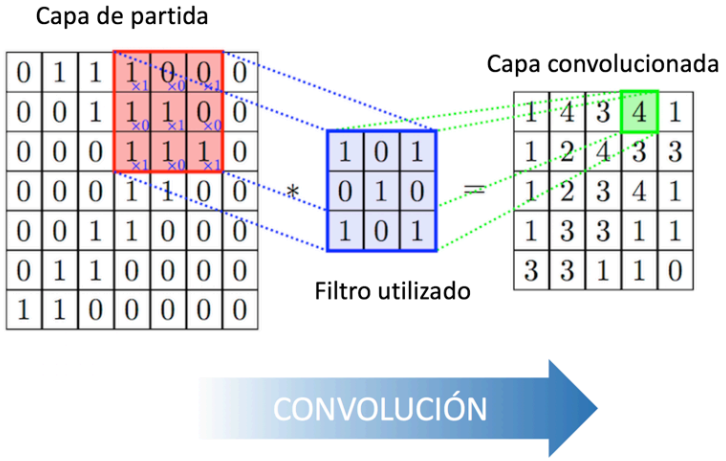


Ilustración 9. Funcionamiento de la capa convolucional. Imagen extraída de http://www.diegocalvo.es/red-neuronal-convolucional/

Para cada una de estas muestras se aplica un Kernel o Filtro y el resultado se trasladará a una posición de la capa de salida. Siguiendo el ejemplo de la Ilustración 9 se aprecia que el 4 es el valor de unos que coinciden después de multiplicar la ventana marcada con el Kernel o filtro seleccionado.

Se puede apreciar como de una entrada representada por una matriz de 7x7 al aplicar el proceso de convolución, se obtiene una salida de 5x5. De esta manera tenemos un mapa de características y a su vez estamos simplificando los datos representados.

Normalmente la capa convolucional suele trabajar con tensores 3D con el formato siguiente: (alto, ancho, profundidad del color). En imágenes RGB la profundidad de color contiene un vector de tres posiciones, una para cada componente de color.

La capa convolucional admite una serie de parámetros como pueden ser la dimensión de la ventana flotante, el desplazamiento que tiene dicha ventana o *stride* y el *padding*. Éstos últimos se verán más adelante en esta propia sección.

Capa Pooling

Las capas de pooling suelen estar concatenadas junto con una capa convolucional. Siendo la entrada de la capa pooling la salida de la capa convolucional.

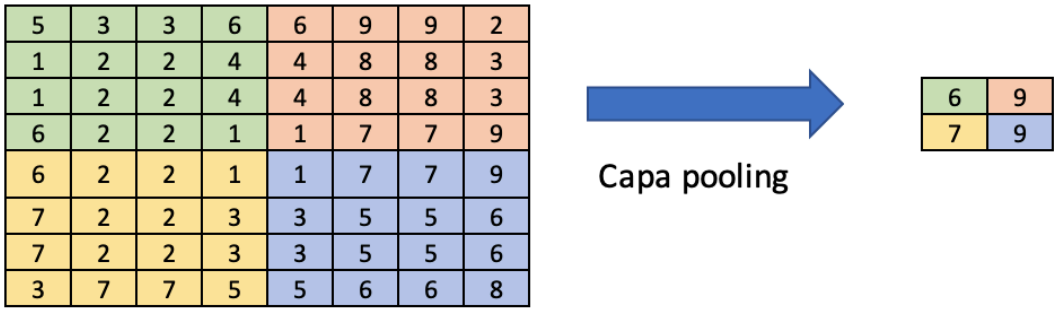


Ilustración 10. Ejemplo de cómo trabaja una capa max-pooling.

En la Ilustración 10 se muestra el funcionamiento de una capa pooling con un valor de venta 4x4. Esta ventana define la dimensión de la matriz a examinar. Al tener una entrada de 8x8 y una capa pooling de 4x4, se examinará 4 áreas definidas por cuatro colores. Dando así, una salida de 2x2.

Para definir que valores rellenan la capa de salida se debe definir un modo para la capa pooling. En el ejemplo de la Ilustración 10 se ha establecido un modo con max-pooling, quiere decir que de cada área examinada se quedará con el valor máximo. Existen otros modos como *average-pooling* que calculará el valor promedio de cada área.

Otros Parámetros de una red convolucional

Adicionalmente existen otros parámetros que deben ser mencionados como pueden ser el *stride* y el *padding*.

Stride

La capa convolucional va examinando pequeños patrones y desplazando esa ventana de izquierda a derecha y de arriba abajo. Por defecto, el valor de desplazamiento o *stride* es 1, pero es un parámetro ajustable y que afectará directamente a la dimensión de salida de la propia capa convolucional.

Padding

Al aplicar una capa de convolución la salida de ésta suele verse comprimida en mayor o menor medida, dependiendo de la dimensión de la ventana flotante y del stride definidos. En ocasiones interesa que la dimensión de la capa de convolución no se vea reducida y sea de la misma dimensión que la entrada de la propia capa. Para esta finalidad existe el parámetro padding cuyo objetivo es rodear de 0 (ceros) los valores de la imagen para

obtener una salida del mismo tamaño. Este comportamiento recibe el nombre de '*zero padding*'. En Keras el padding puede recibir los siguientes valores:

- *Same*, indica a la capa de convolucional que la salida debe ser de iguales dimensiones que la entrada, de esta forma rellenará automáticamente con los 0 (ceros) necesarios para alcanzar dicho objetivo.
- Valid, es el valor predeterminado en Keras. Indica que el padding esta desactivado.

Otras capas interesantes

Keras tiene soporte para múltiples capas divididas en categorías: Core layers, Pooling layers, Noise layers, etc. Se ha querido hacer hincapié sobre dos capas de la categoría Core layers que han sido utilizadas en la construcción de las CNNs de este trabajo: *Dropout* y *Flatten*.

Dropout

La capa Dropout [24] tiene como objetivo desactivar un número aleatorio de neuronas en cada iteración durante el entrenamiento de la red neuronal. Así se reduce el posible sobreajuste que se pueda experimentar. El argumento facilitado indica el *rate* o porcentaje para que una neurona permanezca activa en la iteración en curso. Los valores admitidos van de 0 a 1.

Flatten

La capa Flatten [25] tiene como objetivo aplanar la entrada de la capa, convirtiéndola en un vector de una dimensión. Se suele aplicar antes de la capa de salida.

Image Data Generator

Contar con un dataset con una gran cantidad de imágenes tomadas desde diferentes distancias, ángulos y condiciones de luz sería ideal para entrenar cualquier CNN. De esta forma la CNN se entrenaría con múltiples imágenes para cada categoría, elevando así el porcentaje de precisión obtenido.

Aunque sería ideal disponer con un dataset de estas características, el trabajo que implica tomar todas y cada una de las fotografías, el espació en disco y memoria que ocuparían harían que las fases de entrenamiento y test fueran tareas mucho más engorrosas.

Afortunadamente Keras cuenta con un generador dinámico de imágenes *Image Data Generator (IDG)* [26]. Esté tomará un subconjunto del montante total de entrenamiento y les aplicará una sería de distorsiones a las imágenes originales para obtener un conjunto de imágenes más enriquecido.

IGD admite múltiples argumentos de entrada [26]. Sin embargo, se tratarán los usados en el desarrollo de esta tesis:

- Width_shift_range, define un valor para distorsionar el ancho de la imagen.
- Height_shift_range, define un valor para distorsionar el alto de la imagen.

- Zoom_range, define un valor para distorsionar el zoom aplicado a la imagen.
- Shear_range, define un valor para distorsionar el ángulo de corte aplicado a la imagen.
- Rotation_range, define un valor para distorsionar el ángulo de rotación a la imagen.

Para utilizar el IGD generado, éste se deberá pasar como argumento al método fit_generator() del modelo de Keras que se haya moldeado.

5. Construcción de las redes neuronales

En este punto el lector ya conoce los conjuntos de datos a utilizar: dos para la clasificación de señales, '*German Traffic Sign Detection Benchmark*' [16] y '*Belgium TS Dataset*' [17]. Un conjunto de datos [21] para el problema de la detección de señales. También conoce las funciones de activación, optimizadores y valores de parámetros óptimos para cada red neuronal. Por lo tanto, se está en posición de empezar a crear los modelos para las diferentes redes neuronales utilizadas en este trabajo.

Siguiendo los pasos de Jordi Torres y su libro Deep Learning [3] se van a crear unas primeras redes densamente conectadas para poner en práctica los puntos vistos en el apartado anterior.

5.1. Redes neuronales densamente conectadas

Las redes neuronales densamente conectadas son las redes más básicas que se pueden generar con Keras. Su principal característica es que todas las neuronas de una capa se encuentran interconectadas con todas las neuronas de la capa siguiente.

Debido a esta característica se pueden entrenar redes neuronales de manera muy sencilla. Como primera aproximación se realizarán dos redes densamente conectadas que clasificarán los dataset '*German Traffic Sign Detection Benchmark*' [16] y '*Belgium TS Dataset*' [17]. La configuración de ambas redes se adjunta en la Tabla 5.

#	Dataset	F. Activación	Optimizador	Epoch	Batch_size
1	BelgiumTS	selu	rmsprop	20	32
2	German	selu	rmsprop	20	32
3	BelgiumTS	selu	rmsprop	30	32
4	German	selu	rmsprop	30	32

Tabla 5. Configuración de las redes densamente conectadas

Estas primeras redes neuronales cuentan con tan solo dos capas. La primera de ellas contará con la función de activación establecida, selu. La entrada a la capa será un tensor de una dimensión de 4096 posiciones (64x64). La salida contará con el número de categorías disponibles por cada dataset. La segunda capa, la capa de salida, contará con una función de activación *softmax*. De esta forma la salida será un vector de N posiciones. Siendo N el número de categorías donde se puede clasificar. El vector almacenará en cada posición la probabilidad de pertenecer a cada una de las categorías.

A la hora de compilar el modelo se debe de facilitar una función de perdida, un optimizador y unas métricas. El optimizador queda fijado por la búsqueda de los hiperparámetros realizada en el apartado anterior. Sin embargo, la función de perdida se fija en *categorical_crossentropy* y la métrica utilizada para monitorizar el proceso de aprendizaje de la red será su propia precisión.

Las implementaciones de las redes se encuentran disponibles en el punto 12.4, para la red que usa el dataset de *BelgiumTS* y 12.5 para la red configurada con el dataset *German*. En la Tabla 6 se adjunta un resumen de los resultados obtenidos.

#	Dataset	Pérdida	Precisión	Tiempo por epoch (s)	Tiempo total aprox. (s)
1	BelgiumTS	0,61041733	0,87301587	1	20
2	German	13,7990061	0,03808393	2	40
3	BelgiumTS	0,67986874	0,87420635	1	30
4	German	14,1315918	0,03689628	1	30

Tabla 6. Resultado de las redes densamente conectadas

Analizando los resultados podemos ver como el coste computacional de entrenar este tipo de redes es escaso. En ningún caso se ha excedido del minutó. Por otro lado, se observa como la red entrenada para el dataset BelgiumTS ha obtenido una precisión un poco superior del 87,30%. Un resultado bastante bueno teniendo en cuenta: Uno, se trata de una red densamente conectada y dos, el coste computacional empleado. Sin embargo, la red entrenada con el dataset German, obtiene una precisión del 3,8% en el mejor de los casos.

Existe una disparidad muy elevada entre ambos valores. Es cierto que el dataset BelgiumTS trabaja con una resolución de imágenes de 64x64 píxeles mientras que el dataset German lo hace con una resolución de tan solo 32x32 píxeles. Queda pendiente para futuras vías de investigación, ver punto 7.1 para más información, comprobar si la baja resolución de las imágenes del dataset German es la razón de unos resultados tan bajos. A pesar de eso, se mantuvo el conjunto de datos para su estudio con una red neuronal convolucional.

5.2. Redes neuronales convolucionales

Para alcanzar el objetivo propuesto en este trabajo se ha entrenado a tres CNNs. Una con el objetivo de detectar las posibles señales de tráfico, CNN de detección. Otras dos con el objetivo de clasificar las señales de tráfico detectadas. Cada una de ellas ha sido entrenada con un conjunto de datos diferente. *BelgiumTS* y *German*.

CNN Detección

Se han entrenado 4 modelos de la CNN de detección. Todos ellos con los mismos valores para la función de activación, optimizador, epoch y batch_size. La configuración empleada se detalla en la Tabla 7. Se ha diferenciado las redes entrenadas haciendo uso del generador dinámico de imágenes de Keras y de si se encontraba habilitada o no la interfaz GPU.

#	Dataset	Input	Tipo	batch_size	epoch	F. Activación	Optimizador	GPU	Image Data Generator
6	Detector	64x64	Convolucional	64	25	elu	adam	NO	NO
7	Detector	64x64	Convolucional	64	25	elu	adam	NO	SI

8	Detector	64x64	Convolucional	64	25	elu	adam	SI	NO
9	Detector	64x64	Convolucional	64	25	elu	adam	SI	SI

Tabla 7. Configuración de la CNN de detección.

Para la CNN de detección se ha usado un modelo que contiene 13 capas. En la Tabla 8 se adjunto un resumen de las capas usadas.

Capa número	Tipo	Observaciones
1	Conv2D	La primera capa tiene un filtro de 32 y un *Kernel_size* de (5, 5). La función de activación es elu y la entrada del modelo se define como un tensor2D de (64, 64)
2	MaxPoooling2D	Se aplica una capa de Maxpooling de (2, 2)
3	Dropout	Se aplica una capa Dropout con un factor de activación del 20%
4	Conv2D	Se vuelve a aplicar una capa de Convolución, esta vez con un filtro de 64. El Kernel_size se mantiene en (5, 5).
5	MaxPoooling2D	Se aplica una capa de Maxpooling de (2, 2)
6	Dropout	Se aplica una capa Dropout con un factor de activación del 20%
7	Conv2D	Se vuelve a aplicar una capa de Convolución, esta vez con un filtro de 128. El Kernel_size se mantiene en (5, 5).
8	MaxPoooling2D	Se aplica una capa de Maxpooling de (2, 2)
9	Dropout	Se aplica una capa Dropout con un factor de activación del 20%
10	MaxPoooling2D	Se aplica una capa de Maxpooling de (2, 2)
11	Dropout	Se aplica una capa Dropout con un factor de activación del 20%
12	Flatten	Se aplana la entrada para ofrecer una salida de una sola dimensión.
13	Dense	La capa de salida tiene la función de activación softmax con el fin de realizar una clasificación entre las dos categorías disponibles.

Tabla 8. Resumen de las capas de la CNN de detección

Una vez entrenados los cuatro modelos se han obtenido los resultados adjuntos en la Tabla 9

#	Pérdida	Precisión	Tiempo por epoch (s)	Tiempo total aprox. (s)	Fichero Pesos
6	1,980567	0,822762	95	2.375	model_64_64_det.h5
7	3,182902	0,802526	2.800	70.000	model_data_aug_64_64_det.h5
8	2,134391	0,801032	2	50	model_64_64_det_gpu.h5
9	2,394173	0,851419	575	14.375	model_data_aug_64_64_det_gpu.h5

Tabla 9. Resultados de la CNN de detección.

Finalmente, la mejor CNN entrenada alcanza un valor de precisión de 85,14%.

CNN BelgiumTS

Utilizando el conjunto de datos *BelgiumTS*, se ha elaborado un modelo de CNN con la configuración detallada en la Tabla 10. Se ha utilizado los parámetros ideales obtenidos de los experimentos realizados en el punto **¡Error! No se encuentra el origen de la referencia.**. El modelo se ha entrenado cuatro veces. Dos de ellas sin soporte de GPU (ver punto 9 para más información) y otras dos utilizando el generador de imágenes dinámicas de Keras, explicado en el punto 4.5.

#	Dataset	Input	Tipo	batch_size	epoch	F. Activación	Optimizador	GPU	Image Data Generator
1	BelgiumTS	64x64	Convolucional	128	25	selu	rmsprop	NO	NO
2	BelgiumTS	64x64	Convolucional	128	25	selu	rmsprop	NO	SI
3	BelgiumTS	64x64	Convolucional	128	25	selu	rmsprop	SI	NO
4	BelgiumTS	64x64	Convolucional	128	25	selu	rmsprop	SI	SI

Tabla 10. Configuración de la CNN de clasificación: BelgiumTS.

El detalle del modelo implementado es exactamente igual al mostrado en la Tabla 8. Exceptuando la capa de entrada que admite un tensor 3D de (64, 64, 3).

Finalmente se han obtenido unos valores de pérdida y precisión detallados en la Tabla 11.

#	Pérdida	Precisión	Tiempo por epoch (s)	Tiempo total aprox. (s)	Fichero Pesos
1	0,140183019	0,978174603	40	1.000	model_64_64_bel.h5
2	0,29150691	0,981746032	1.390	34.750	model_data_aug_64_64_bel.h5
3	0,26388365	0,952380952	1	25	model_64_64_bel_gpu.h5
4	0,239080064	0,98452381	150	3.750	model_data_aug_64_64_bel_gpu.h5

Tabla 11. Resultados de la CNN de clasificación BelgiumTS.

Lo primero que llama la atención es los 34.750 segundos ~ 10 horas que ha tardado en entrenar una CNN únicamente usando la CPU. Comparando la misma configuración, pero con la interfaz GPU habilitada, se ha tardado 3750 segundos ~ 1hora.

Los valores de precisión obtenidos en las cuatro CNN son unos valores bastante aceptables llegando a un máximo de 98,45% de precisión. También se adjunta el fichero de pesos generados para cargar rápidamente el modelo entrenado a la hora de realizar las pruebas.

CNN German

Utilizando el conjunto de datos *German*, se ha elaborado un modelo de CNN con la configuración detallada en la Tabla 12. Al tratarse de una CNN de clasificación se ha copiado los valores de parámetros de la CNN BelgiumTS. El modelo se ha entrenado cuatro veces. Dos de ellas sin

soporte de GPU (ver punto 9 para más información) y otras dos utilizando el generador de imágenes dinámicas de Keras, explicado en el punto 4.5.

#	Dataset	Input	Tipo	batch_size	epoch	F. Activación	Optimizador	GPU	Image Data Generator
10	German	32x32	Convolucional	128	25	selu	rmsprop	NO	NO
11	German	32x32	Convolucional	128	25	selu	rmsprop	NO	SI
13	German	32x32	Convolucional	128	25	selu	rmsprop	SI	NO
14	German	32x32	Convolucional	128	25	selu	rmsprop	SI	SI

Tabla 12. Configuración de la CNN de clasificación: German.

Debido a la reducida resolución de las imágenes del conjunto German, el modelo de red neuronal se ha visto simplificando en base a las dos anteriores CNN. En la se muestra el detalle del modelo empleado para el conjunto German.

Capa número	Tipo	Observaciones
1	Conv2D	La primera capa tiene un filtro de 32 y un *Kernel_size* de (5, 5). La función de activación es elu y la entrada del modelo se define como un tensor 3D de (32, 32, 3)
2	MaxPoooling2D	Se aplica una capa de Maxpooling de (2, 2)
3	Dropout	Se aplica una capa Dropout con un factor de activación del 20%
4	Conv2D	Se vuelve a aplicar una capa de Convolución, esta vez con un filtro de 64. El Kernel_size se mantiene en (5, 5).
5	MaxPoooling2D	Se aplica una capa de Maxpooling de (2, 2)
6	Dropout	Se aplica una capa Dropout con un factor de activación del 20%
7	MaxPoooling2D	Se aplica una capa de Maxpooling de (2, 2)
8	Dropout	Se aplica una capa Dropout con un factor de activación del 20%
9	MaxPoooling2D	Se aplica una capa de Maxpooling de (2, 2)
10	Dense	La capa de salida tiene la función de activación softmax con el fin de realizar una clasificación entre las categorías disponibles.

Tabla 13. Resumen de las capas del modelo para la CNN clasificatoria con el dataset German.

Se ha obtenido unos valores de precisión detallados en la Tabla 14.

#	Pérdida	Precisión	Tiempo por epoch (s)	Tiempo total aprox. (s)	Fichero Pesos
10	15,09903035	0,037371338	72	1800	model_32_32_ger.h5
11	15,2334593	0,037925574	1500	37500	model_data_aug_32_32_ger.h5
13	15,11129743	0,037212985	2	50	model_32_32_ger_gpu.h5
14	15,27319798	0,037371338	575	14375	model_data_aug_32_32_ger_gpu.h5

Tabla 14. Resultados de la CNN de clasificación German.

Al igual que pasaba con la CNN entrenada con el dataset *BelgiumTS*, se aprecia la reducción de tiempo empleado habilitando la interfaz GPU. Sin embargo, los valores de precisión no han mejorado respecto a la red densamente conectada entrenada con el mismo conjunto de datos. Se ha obtenido un valor de precisión de tan solo 3,7% un valor insuficiente y muy alejado de los valores obtenidos con la CNN entrenada con *BelgiumTS*. Observando estos resultados, se remarca especial atención al punto 7.1 para intentar revertir estos valores.

6. Ejemplos de funcionamiento de las CNNs

Una vez construidas todas las CNNs se va a realizar una batería de pruebas para diagnosticar que las CNNs funcionan correctamente. Para ello, se ha utilizado una pequeña colección de imágenes que no pertenecen a ningún conjunto de datos vistos anteriormente. Se trata de imágenes extraídas de Google Street View del pueblo de Cardedeu. Todas ellas tienen una resolución de 360 píxeles de ancho por 270 píxeles de alto. En la Ilustración 11 se muestran las imágenes.

Ilustración 11. Imágenes de pruebas del pueblo de Cardedeu extraídas de Google Street View.

6.1. CNN Detección de señales

El primer test por realizar es comprobar la efectividad del detector de imágenes entrenado. Debido a que nuestra CNN debe recibir una entrada de 64x64 píxeles y las imágenes tienen una resolución de 360x270 píxeles. Lo primero que se debe de hacer es una función en Python que vaya recorriendo de izquierda a derecha y de arriba a bajo cada una de las imágenes de prueba. La dimensión de esta ventana flotante será de 64x64. Así se podrá comprobar en que zonas de la imagen existe una señal de tráfico.

Al tratarse de imágenes de 360x270 píxeles, trabajar con una ventana de 64x64 píxeles y configurar un desplazamiento de un píxel para cada iteración. Se obtiene un total de 60.976 celdas por cada imagen.

La CNN empleada para la detección de tráfico se ha cargado a partir del fichero de pesos generados en el entrenamiento de la CNN detallada en el punto 5.2, más concretamente la red marcada con el identificador #9. La cual ha sido entrenada con soporte para GPU y con el generador dinámico de imágenes de Keras activado.

En cada una de las 60.976 iteraciones, se aplicará la CNN de detección de señales de tráfico, en caso de que el detector responda de manera afirmativa, se pintará un cuadrado de 5x5 de color rojo en el centro de la ventana que haya dado la lectura positiva. Una vez ejecutado la prueba obtenemos los resultados adjuntos en la Ilustración 12.

Ilustración 12. Resultados de la prueba a la CNN de detección de señales de tráfico

Se puede observar como la CNN genera algunos falsos positivos. En términos generales identifica correctamente la ubicación de las señales de tráfico, dando por válido el entrenamiento de la CNN, detallada en el punto 5.2.

6.2. CNN Clasificación de señales

La segunda prueba consiste en testear la CNN de clasificación. Para ello se ha recortado exactamente las señales de tráfico de las imágenes originales. Los recortes tienen una dimensión de 64x64 píxeles. Se pueden ver los recortes en la Ilustración 13.

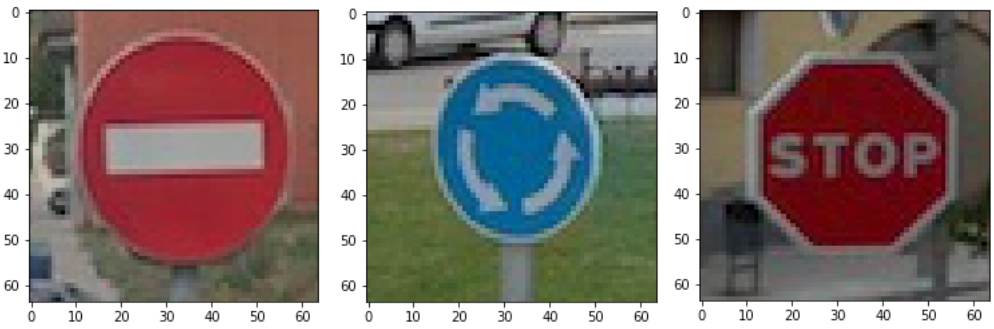


Ilustración 13. Recortes de las señales de tráfico.

Se han usado dos CNNs, correspondientes a cada uno de los conjuntos de datos usados: *BelgiumTS* y *German*. La primera se ha cargado a partir del fichero de pesos generado por la CNN detallada en el punto 5.2, se trata de la CNN identificada con #4. Más concretamente se ha entrenado haciendo uso del conjunto de datos *BelgiumTS*, '*selu*' como función de activación, '*rmsprop*' como optimizador y 128 y 25 como valores de *batch_size* y *epoch* respectivamente. Además, se ha entrenado con el soporte de GPU activado y con el generador de imágenes dinámicas de Keras habilitado.

La segunda CNN se ha cargado a partir del fichero de pesos generado por la CNN identificada con #14. Ha sido entrenada con el dataset *German* y su configuración es homónima a la anterior CNN. La única diferencia consta del dataset empleado en su entrenamiento.

Una vez ejecutada la prueba se han obtenido los resultados de la Ilustración 14. La implementación de la prueba se encuentra en el punto 12.10

```
print("Belgium:")
class_result_bel = model_bel.predict_classes(images_test_bel)
for result_bel in class_result_bel:
    print("Clase resultante: {} y título: {}". format(result_bel, titles_bel[result_bel]))


print("Germany:")
class_result_ger = model_ger.predict_classes(images_test_ger)
for result_ger in class_result_ger:
    print("Clase resultante: {} y título: {}". format(result_ger, titles_ger[result_ger]))
```

```
Belgium:
Clase resultante: 22 y título: ['22', 'C1']
Clase resultante: 37 y título: ['37', 'D5']
Clase resultante: 21 y título: ['21', 'B5']
Germany:
Clase resultante: 17 y título: ['17', 'no entry (other)']
Clase resultante: 40 y título: ['40', 'roundabout (mandatory)']
Clase resultante: 14 y título: ['14', 'stop (other)']
```

Ilustración 14. Resultados de la prueba aplicado a la CNN de clasificación.

Se puede apreciar como ambas CNN han clasificado las señales de tráfico con un 100% de acierto. Sorprende que la CNN con el dataset *German* haya clasificado de manera correcta las señales, ya que tan solo tenía un valor de precisión del 3,7%.

En la Ilustración 15 se adjuntan las señales C1, D5 y B5 correspondientes al dataset *BelgiumTS.*

Ilustración 15. Señales C1, D5 y B5 extraídas del dataset BelgiumTS.

En la Ilustración 16 se adjuntan las señales 17, 40 y 14 pertenecientes al dataset *German.*

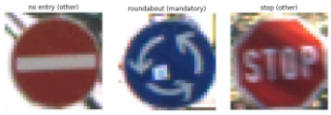


Ilustración 16. Señales 17, 40 y 14 extraídas del dataset German.

6.3. CNN Clasificación de señales – Imágenes completas

Se ha realizado otra prueba sobre las CNNs de clasificación. En lugar de facilitar la imagen de la señal correspondientemente recortada, se facilitará la imagen completa de la escena. Esto tiene como finalidad demostrar la necesidad de implementar un detector de señales primero y un clasificador después.

La configuración de las CNN es exactamente la misma que en el punto 6.2 y la implementación de la prueba se encuentra adjuntada en el apartado 12.11. Una vez lanzada la prueba se han obtenido los resultados que se muestran en la Ilustración 17.

```
print("Belgium:")
class_result_bel = model_bel.predict_classes(images_test_bel)
for result_bel in class_result_bel:
    print("Clase resultante: {} y título: {}". format(result_bel, titles_bel[result_bel]))


print("Germany:")
class_result_ger = model_ger.predict_classes(images_test_ger)
for result_ger in class_result_ger:
    print("Clase resultante: {} y título: {}". format(result_ger, titles_ger[result_ger]))
```

```
Belgium:
Clase resultante: 19 y título: ['19', 'B1']
Clase resultante: 7 y título: ['7', 'A23']
Clase resultante: 24 y título: ['24', 'C21']
Germany:
Clase resultante: 5 y título: ['5', 'speed limit 80 (prohibitory)']
Clase resultante: 12 y título: ['12', 'priority road (other)']
Clase resultante: 10 y título: ['10', 'no overtaking (trucks) (prohibitory)']
```

Ilustración 17. Resultados de la prueba a las CNNs de clasificación con imágenes completas.

Se observa como en este caso el porcentaje de acierto se reduce al 0% para ambas CNNs. Por lo tanto, no se puede aplicar directamente la CNN de clasificación sobre una imagen. Queda demostrada la necesidad de diseñar un esquema que recorra la imagen para detectar primero donde se ubican las señales y posteriormente aplicar el clasificador.

6.4. Ejemplo del sistema completo

Se ha diseñado un script combina ambas CNNs para trabajar de manera colaborativa y poder detectar y clasificar una señal de tráfico a partir de una imagen estática.

El guion de la implementación se trata en:

- Cargar una imagen
- Descomponer la imagen en celdas de 64x64. Con un avance de un píxel las celdas irán recorriendo la imagen de izquierda a derecha y de arriba a bajo.
- Cada celda se facilita a la CNN de detección de señales.
- Si la respuesta es negativa, se saltará a la siguiente celda. En caso de ser positiva, la celda se pasará a la CNN de clasificación de señales de tráfico.
- El resultado de la CNN de clasificación se almacena en un vector temporal.
- Una vez recorridas todas las celdas de una imagen, se extrae la moda del vector temporal. Siendo este valor el resultado de la clasificación de la señal.

La configuración de las CNNs usadas en esta prueba será la misma que se ha visto en el apartado 6.1 y 6.2. Respecto a la clasificación solo se ha usado la CNN entrenada con el conjunto de datos *BelgiumTS*.

El código de la prueba se encuentra adjunto en el punto 12.12. En la Ilustración 18 se muestran los resultados obtenidos.

```
signs = []
for image_path in images_test_path:
    image_test = get_normalized_image(image_path)
    image_tiles = get_tiles(image_test, IMG_SHAPE, 1)
    signs_aux = []
    for tile in image_tiles:
        np_tile = np.expand_dims(tile, axis=0)
        result = model_detector.predict_classes(np_tile)
        if result == 1:
            result_sign = model_classificator.predict_classes(np_tile)
            signs_aux.append(result_sign[0])
    print("Tiles: {} - valid tiles: {} Moda: {} Sign {}".format(len(image_tiles), len(signs_aux), mode(signs_aux), titl
    signs.append(signs_aux)
```

```
Tiles: 60976 - valid tiles: 7588 Moda: 22 Sign ['22', 'C1']
Tiles: 60976 - valid tiles: 5750 Moda: 35 Sign ['35', 'D1b']
Tiles: 61182 - valid tiles: 4048 Moda: 22 Sign ['22', 'C1']
```

Ilustración 18. Resultados de la prueba del sistema de CNN detectora más la CNN de clasificación.

Se puede apreciar que tan solo se ha detectado y clasificado de manera correcta una señal. La primera como C1. Se puede ver las señales detectadas en la Ilustración 19.

Ilustración 19. Señales detectadas y clasificadas por el sistema.

Analizando y comparando los resultados más detalladamente se alcanzan las siguientes conclusiones:

- Las señales detectadas no coinciden con las señales detectadas en la prueba con imágenes completas. Por lo tanto, podemos afirmar que el detector de señales si que ha detectado la ubicación de las señales correctamente.
- Estudiando los resultados obtenidos y el cómo se obtienen, se aprecia que las señales detectadas se consiguen a raíz de hacer la moda de todas las lecturas positivas realizadas por el detector. Esto quiere decir que los falsos positivos que se han visto en el punto 6.1 están desvirtuando la acción del clasificador. Para dar más veracidad a esta teoría se aprecia como en caso obtener una imagen clara de las señales de tráfico el clasificador realiza la clasificación con un 100% de precisión. Ver punto 6.2
- Recorrer las 60. 976 celdas aplicando la CNN de detección a cada una de ellas, es un sistema altamente ineficiente ya que requiere mucho tiempo computacional.

La realización de las pruebas llevadas a cabo han destapado la necesidad de continuar trabajando en el sistema, solventando algunas carencias actuales.

7. Futuras vías de investigación

El tema trabajado a lo largo de esta tesis es bastante amplio, requiere una gran cantidad de tiempo para afrontar los diferentes entrenamientos y sus correspondientes pruebas. Sobre todo, en tiempos de computación a la hora de entrenar nuevas CNNs.

Debido a que esta tesis tiene un tiempo limitado, se debe de estimar un corte en los análisis y desarrollos de las diferentes pruebas realizadas. Debido a eso, este apartado analizará los que serían los siguientes pasos que se deberían aplicar en caso de disponer de más tiempo.

7.1. Mejorar resultados de las redes neuronales con dataset alemán

A lo largo del proyecto se ha trabajado con dos dataset: *BelgiumTS* y *German*. Ambos están más detallados en el apartado 3.4. Durante el desarrollo se ha experimentado como mientras el dataset *BelgiumTS* obtenía una precisión del 98,45%, el conjunto *German* no ha podido entrenar una CNN que alcance ni tan siquiera el 4%. A pesar de contar con un conjunto más amplio de imágenes de entrenamiento y test. Sin embargo, sí es cierto que la resolución de sus imágenes es menor que la del conjunto *BelgiumTS*.

Este hecho marcaría el primer punto en el que se debe seguir trabajando: Realizar más pruebas, cambiando hiperparámetros de la CNN para poder alcanzar unos valores más afines a la CNN entrenada con el dataset BelgiumTS.

7.2. Mejorar resultados del sistema

A pesar de contar con una CNN clasificatoria de precisión 98,45%. Los experimentos realizados en el punto **¡Error! No se encuentra el origen de la referencia.** no han tenido un número de aciertos suficientes para clasificarlo como un sistema válido. De 3 imágenes tan solo se ha clasificado una de manera satisfactoria. (33%)

La revisión de este punto sería crítica, para poder conocer cuáles son las malas configuraciones que estén afectando de manera tan significativa y negativamente a los resultados finales.

Una posible nueva reestructuración del sistema para evitar la cantidad de falsos positivos diagnosticados por la CNN detectora, que sin suda, están afectando al resultado y rendimiento global. Por ejemplo, se podría dividir la foto original en áreas más grandes e ir ampliado la zona de aquellas zonas que den un resultado positivo, hasta intentar aislar las áreas donde se ubica una señal. Esta implementación también solucionaría el problema visto en el siguiente punto.

Otra posible implementación sería contemplar donde hubiera más detecciones de señales, extraer la bounding box y facilitar esa área a la CNN de clasificación. De esta manera se reduciría enormemente el tiempo de proceso.

También se puede modificar los dataset BelgiumTS y German para ampliar sus categorías con una nueva señal. Esta señal haría referencia "No es señal de tráfico" así propiamente el clasificador también ignoraría los falsos positivos que se produzcan en la CNN de detección.

7.3. Imágenes con señales simultáneas

Como se ha podido apreciar en el experimento del punto **¡Error! No se encuentra el origen de la referencia.**. Para obtener un resultado se tienen en cuenta todas las clasificaciones realizadas y posteriormente se examina el valor más repetido, la moda. De esta manera la señal más repetida es la que el sistema identificará. Es un sistema que requiere de una revisión. Primero, porque obliga a activar la segunda CNN en múltiples ocasiones, lo que penaliza significativamente el tiempo de respuesta del sistema. Segundo, en caso de disponer de una imagen con más de una señal, el sistema no devolverá una respuesta con dos señales.

Por eso se debería de replantear esta combinación de CNNs para poder detectar múltiples señales de tráfico en una única imagen y reducir el tiempo de respuesta.

7.4. Migración hacía dispositivos móviles iOS

Otra vía de investigación y la menos prioritaria, sería poder migrar los modelos de Keras hacía dispositivos móviles iOS. A partir del lanzamiento de iOS 11. Apple introdujo la librería de Machine Learning: Core ML [28]. Revisando su documentación [29] se puede ver, Ilustración 20, el flujo que sigue un hipotético modelo dentro del ecosistema Apple.

Ilustración 20: Imagen extraída de https://developer.apple.com/documentation/coreml#overview

Además, la propia documentación de Apple [29] explica como migrar un modelo previamente entrenado para ser usado con Core ML. En la Ilustración 21 se muestra los frameworks soportados para los diferentes tipo de modelos admitidos por Core ML. Se puede apreciar como las redes neuronales entrenadas con Keras son compatibles a partir de su version 1.2.2

Model type	Supported models	Supported frameworks
Neural networks	Feedforward, convolutional, recurrent	Caffe v1 Keras 1.2.2+
Tree ensembles	Random forests, boosted trees, decision trees	scikit-learn 0.18 XGBoost 0.6
Support vector machines	Scalar regression, multiclass classification	scikit-learn 0.18 LIBSVM 3.22
Generalized linear models	Linear regression, logistic regression	scikit-learn 0.18
Feature engineering	Sparse vectorization, dense vectorization, categorical processing	scikit-learn 0.18
Pipeline models	Sequentially chained models	scikit-learn 0.18

Ilustración 21: Imagen extraída de https://developer.apple.com/documentation/coreml/converting_trained_models_to_core_ml

Debido a esta aparente facilidad para migrar modelos, sería interesante realizar una aplicación para dispositivos móviles capaz de detectar y clasificar señales de tráfico en tiempo real haciendo uso de las cámaras equipadas en los dispositivos Apple.

8. Problemas surgidos

Durante la realización de este trabajo, han surgido diversos problemas. Los más relevantes se detallan a continuación:

8.1. Macbook: Problema con GPU.

El entrenamiento de una red neuronal es una tarea que requiere gran capacidad computacional. Con el fin de reducir este tiempo, se puede aprovechar la actual potencia de los sistemas gráficos equipados en los ordenadores. La idea es paralelizar ciertos cálculos matemáticos y realizarlos en la GPU, liberando de esta manera la carga de la CPU. Keras y sus backends soportan el uso de GPUs mediante la tecnología CUDA propietaria de Nvidia.

Como se describe en el punto 11.2 para la realización de esta tesis se han usado dos equipos: Uno con una tarjeta gráfica AMD-ATI, por lo que no se podía hacer uso de CUDA y un segundo equipo, con una tarjeta Nvidia Geforce GT 650M [25]. Se esperaba usar este equipo para agilizar los cálculos de las diferentes redes neuronales.

El primer contratiempo se obtuvo a la hora de instalar los drivers CUDA y la extensión de Tensorflow para habilitar el uso de la GPU. Bajo el sistema operativo Mac OS High Sierra, fue imposible hacer correr Keras con Tensorflow-GPU. Actualizando el sistema a Mac OS Mojave, directamente fue imposible instalar los drivers de CUDA, ya que Apple no ha liberado los drivers. Por lo tanto, es imposible usar CUDA bajo el último sistema de Apple.

Para solucionar este incidente, se optó por instalar de manera nativa el sistema operativo Ubuntu 18.04 [26]. En él, se pudo instalar los drivers de CUDA y hacer correr Keras con el backend Tensorflow-GPU. Una vez todo estaba configurado se procedió a entrenar la primera red neuronal.

En ese momento se descubrió que cada chip gráfico usado por Nvidia tiene una puntuación llamada capacidad computacional [27]. El mínimo necesario para ejecutar Tensorflow-GPU es de 3.5 y el chip gráfico utilizado en el equipo está valorado con una puntuación de 3.0. Como se adjunta en la Ilustración 22 se muestra un extracto de los diferentes productos de Nvidia y sus puntuaciones.

GeForce GT 755M	3.0
GeForce GT 750M	3.0
GeForce GT 650M	3.0
GeForce GT 745M	3.0
GeForce GT 645M	3.0

Ilustración 22: Resumen de la capacidad computacional CUDA de la GT 650M.

Debido a esto, se desistió ejecutar Keras con ningún soporte para GPU habilitando, entrenando todas las redes neuronales usando única y exclusivamente la CPU, lo que alargo el tiempo de entrenamiento en redes neuronales convolucionales.

8.2. Lentitud a la hora de diagnosticar señales de tráfico.

Otro problema surgido, hace referencia al rendimiento. Durante la prueba realizada combinando las dos CNN (detector y clasificador) se pudo apreciar como para detectar y clasificar una imagen de tan solo 260 x 320 pixeles el tiempo invertido es superior al deseado. Llegando a alcanzar tiempos superiores al minuto. Al tratarse de imágenes de tan poca resolución lo ideal sería detectar y clasificar señales de manera instantánea.

Este problema junto al punto 7.3, hace necesario una revisión del planteamiento llevado a cabo en este trabajo.

9. Google Colaboratory

A raíz del problema descrito en el punto 8.1, se buscó una posible solución para poder entrenar redes neuronales haciendo uso de una GPU. Con fin reducir drásticamente el tiempo empleado en entrenamiento de redes neuronales. Casi finalizado esta tesis se descubrió Google Colaboratory [28]. Tal como se describe en su propia web:

> *Colaboratory es un entorno gratuito de Jupyter Notebook que no requiere configuración y que se ejecuta completamente en la nube.*

Aparte de la comodidad que ofrece poder ejecutar los notebooks sin necesidad de tener un entorno configurado, Google Colaboratory ofrece la posibilidad de usar una GPU en el entorno remoto. En la Ilustración 23 se puede ver la interfaz usada por Tensorflow al ejecutarse en la plataforma de Google Colaboratory. En la última línea podemos ver como la interfaz GPU habilitada tiene una puntuación computacional de 7.5, actualmente el máximo de puntuación.

```
[ ]  # Mostramos información de CPU & GPU

     from tensorflow.python.client import device_lib
     print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 6591234050081998241
, name: "/device:XLA_CPU:0"
device_type: "XLA_CPU"
memory_limit: 17179869184
locality {
}
incarnation: 16894059133533431556
physical_device_desc: "device: XLA_CPU device"
, name: "/device:XLA_GPU:0"
device_type: "XLA_GPU"
memory_limit: 17179869184
locality {
}
incarnation: 2859374630618488523
physical_device_desc: "device: XLA_GPU device"
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 14800692839
locality {
  bus_id: 1
  links {
  }
}
incarnation: 6888324098725513013
physical_device_desc: "device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5"
]
```

Ilustración 23. Resumen de la interfaz que ofrece Google Colaboratory

Haciendo uso de la interfaz facilitada se ha replicado el entrenamiento de las CNN usadas en este trabajo. En la Tabla 15 se puede apreciar un estudio comparativo de los tiempos empleados para entrenar cada una de las CNNs.

#	CNN	Dataset	Image Data Generator	CPU		CPU + GPU		Diferencia (s)	Speed Up
				Tiempo por epoch (s)	Tiempo total Aprox (s)	Tiempo por epoch (s)	Tiempo total Aprox (s)		
1	Clasificación	BelgiumTS	NO	40	1.000	1	25	975	98%
2	Clasificación	BelgiumTS	SI	1.390	34.750	150	3.750	31.000	89%
3	Clasificación	German	NO	72	1800	2	50	1.750	97%
4	Clasificación	German	SI	1500	37500	575	14375	23.125	62%
5	Detección	Detección	NO	95	2375	2	50	2.325	98%
6	Detección	Detección	SI	2800	70000	575	14375	55.625	79%

Tabla 15. Comparativa de los tiempos empleados en el entrenamiento de las CNN.

Examinando los resultados, es más que evidente la mejora que implica entrenar una red neuronal con una interfaz GPU activada. Haciendo posible entrenar redes mayores, más profundas y de mayor complejidad que por lo contrario sería imposible entrenarlas haciendo uso exclusivo de las CPU actuales.

10. Otras aplicaciones

Existen una gran multitud de librerías destinadas a la clasificación de imágenes. Algunos ejemplos pueden ser Yolo [25], RetinaNet [26] o ImageAI [27] entre otras.

Se ha querido realizar una aproximación a una de estas librerías. Realizando una prueba con ImageAI, se ha desarrollado un pequeño programa en Python, capaz de detectar objetos de un video pregrabado. El código se adjunta en el punto 12.15.

10.1. Test con Video

Se ha grabado un video [28] circulando en un tramo de carretera con tráfico real. Se puede ver un fotograma en la Ilustración 24.

Ilustración 24: Fotograma del video - Test de recorrido

Se ha aplicado el programa generado y se ha obtenido una salida en forma de video [29]. Se adjunta fotograma en la Ilustración 25.

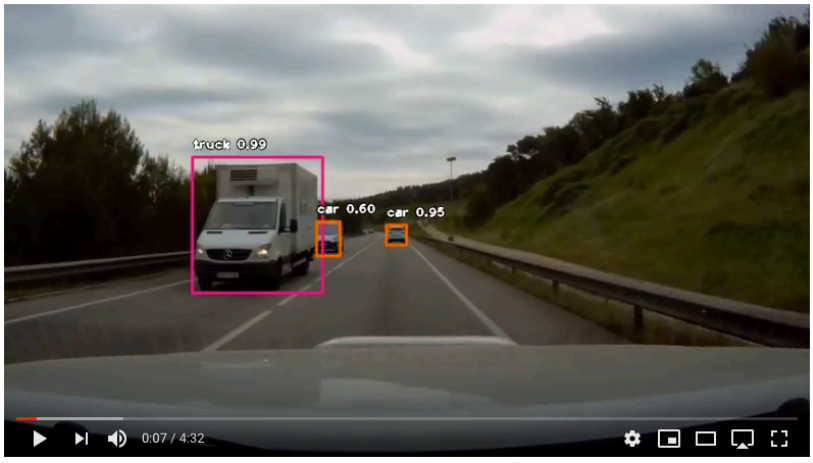


Ilustración 25: Fotograma del video – Test de recorrido con ImageIA

Al realizar el computo del video en una máquina sin una interfaz de GPU válida, el tiempo invertido para analizar un video de 4 minutos y 32 segundos ha sido superior a 2 horas. Quedando nuevamente patente la necesidad de poder disponer de una GPU válida para este tipo de tareas.

11. Anexo 1: Preparación del entorno

11.1. Introducción

Este anexo alberga información sobre el hardware y software usados para la preparación del entorno de desarrollo usado en la realización de este TFM.

11.2. Hardware usado

Para la realización de este TFM se han usado dos equipos informáticos. El primero de ellos, un equipo de sobremesa iMac, con la configuración de la Tabla 16

Nombre de Equipo	iMac
Sistema Operativo	macOS Mojave 10.14.1
Procesador	Intel Core i7 4Ghz
Número de núcleos	4
Cache nivel 2	256 KB. (Por núcleo)
Cache nivel 3	8 MB
Memoria	16 GB
Tarjeta grafica	AMD Radeon R9 M295X
VRAM	4096 MB. (4 GB.)

Tabla 16. Hardware iMac

También se usará un segundo equipo, un portátil Macbook Pro con la configuración de la Tabla 17

Nombre de Equipo	Macbook
Sistema Operativo	macOS High Sierra 10.13.6

Procesador	Intel Core i7 2,3 Ghz
Número de núcleos	4
Cache nivel 2	256 KB. (Por núcleo)
Cache nivel 3	6 MB
Memoria	16 GB
Tarjeta grafica	nVidia GeForce GT 650M
VRAM	512MB

Tabla 17. Hardware Macbook Pro

11.3. Virtualización de entornos

La virtualización de entornos de desarrollo tiene muchas ventajas. Una de ellas, sino la más importante, es la capacidad de aislar el entorno de desarrollo del resto del entorno del ordenador. Esto nos aporta la capacidad de crear diferentes servicios que podrán ser desplegados en multitud de entornos con la certeza de que tendrán un funcionamiento igual al que tuvo el desarrollador cuando lo creo.

Debido a que este TFM tiene un componente bastante alto de investigación y testeo de diferentes librerías, se ha optado por encapsular los entornos de desarrollo. Así podremos aislar nuestro sistema principal de los entornos de ejecución. Recrear rápidamente nuevos entornos con la configuración y librerías ya instaladas.

Actualmente existen diversas herramientas para la creación de entornos virtuales. Una de las más conocidas en la industria es Docker [1]. Así que se ha optado por su uso. También se usará un entorno generado por la herramienta VirtualEnv [2]. Esta última, particularmente la encuentro más sencilla y con una curva de aprendizaje menor que Docker. Por el contrario, únicamente ofrece la posibilidad de virtualizar entornos de desarrollo en Python, en maquinas que corran un sistema UNIX. Adversidad que para el objetivo de este trabajo se ve minimizado ya que únicamente trabajaremos con Python y los equipos usados corren mac OS.

11.4. Librerías necesarias

En ambos entornos de desarrollo se necesita tener instaladas las siguientes librerías.

- Theano

- Tensorflow
- Keras
- Numpy
- Scipy
- Matplotlib
- Ipython
- Pandas
- Sympy
- Nose
- Scikit-image
- Jupyter
- Import_ipynb

11.5. Docker

Docker se trata de una herramienta muy popular entre los desarrolladores, ya que permite la distribución y puesta en marcha de aplicaciones de manera rápida mediante contenedores. Además, estos contenedores aíslan el sistema principal del contenido del propio contenedor. Actualmente Docker cuenta con clientes para Windows, Linux y Mac OS, haciendo de esta manera que los contenedores sean multiplataformas.

En este punto es necesario explicar ciertas nociones básicas de Docker para facilitar al lector la ejecución de este o de otros contenedores en su sistema principal.

Las imágenes en Docker son las entidades más básicas que nos podemos encontrar. Las imágenes son una captura de un sistema preconfigurado. Las imágenes se pueden descargar de Internet donde se pueden encontrar un gran número de imágenes preconfiguradas por los usuarios. En la web de Docker Hub [22] se pueden encontrar un gran número.

También podemos generar imágenes mediante la creación de un fichero DockerFile. Esté archivo se comporta como un script, donde se establece la imagen base a usar, los comandos a ejecutar en la imagen, punto de entrada, mapeos de puertos, etc.

Con el siguiente comando compilaremos el fichero DockerFile creando una imagen con un tag identificativo 'traffic'

```
docker build -t traffic .
```

Ahora se ha creado una imagen que sirve como punto de partida para crear todos los contenedores que se necesiten. Dicha imagen ya tiene instaladas todas las librerías necesarias para la ejecución de este TFM.

Con el fin de facilitar el paso anterior, se ha subido una imagen al hub de docker [23] desde donde se puede descargar con el comando:

```
docker pull agonzalezhidalgo/traffic
```

Para visualizar las imágenes que hay almacenadas en el repositorio local

```
docker images --all
```

Para crear y ejecutar un contendor se usa el comando

```
docker run -p 8888:8888 -t agonzalezhidalgo/traffic
```

El comando -p sirve para enlazar un puerto de la máquina huésped con la máquina anfitrión. En el comando estamos enlazado los puertos 8888 de ambas máquinas. En caso de querer mapear una carpeta del sistema principal con el contenedor que se va a crear, se debe añadir el argumento:

```
-v <carpeta_local>:<carpeta_contenedor>
```

Al ejecutar la imagen, docker creará un contendor automáticamente. Para visualizar los contenedores que disponemos, se puede ejecutar:

```
docker container ls
```

con los argumentos *-a* se muestran todos los contenedores, con *-ps* se muestran los contenedores que actualmente se encuentran en ejecución.

Una vez se ha creado un contenedor, si se quiere volver a ejecutarlo se debe de usar

```
docker start <contenedor_id>
```

11.6. VirtualEnv

VirtualEnv [2] se trata de una herramienta más sencilla que Docker, pero bastante útil y de gran potencial. Ya que nos permite aislar entornos de desarrollo de Python, sobre máquinas que corran un sistema UNIX.

Esta herramienta al ser mucho más liviana que Docker, puede ofrecer en determinados aspectos mejores rendimientos, sobretodo si se tratan de entornos puramente Python y con recursos limitados.

Para automatizar la creación del entorno, se ha escrito un bash script. Una vez ejecutado el script, se dispondrá de un entorno totalmente funcional con el nombre indicado. Para activarlo se deberá ejecutar

```
source ~/<environment_name>/bin/activate
```

Una vez el entorno este activado, se apreciará como a la derecha de la terminal aparecerá entre paréntesis el nombre de nuestro entorno. Se adjunta un ejemplo en la Ilustración 24

Ilustración 26. Ejemplo de entorno de virtualEnv en ejecución

12. Anexo 2: Implementaciones

Este anexo recopila los diferentes códigos fuentes escritos para la realización de este TFM. Adicionalmente se encuentran disponibles en el repositorio de GitHub [36].

12.1. Dockerfile

El contenedor Docker se ha generado con el fichero Dockerfile de la Tabla 18.

```
# Author:   Antonio González Hidalgo (agonzalezhidalgo@uoc.edu)
# TFM:      Traffic signals recognition
# Date:     Jan-2019

# Set basic images
FROM ubuntu:latest

# Set maintainer information
LABEL maintainer="Antonio González (agonzalezhidalgo@uoc.edu)"

# Run a system update, install python3 and pip3
RUN apt-get update && apt-get install -y python3 python3-pip

# Install jupyter, tensorflow, theano, keras and useful libraries
RUN pip3 install jupyter
RUN pip3 install --upgrade tensorflow
RUN pip3 install theano
RUN pip3 install keras
RUN pip3 install numpy scipy matplotlib ipython pandas sympy nose
RUN pip3 install -U scikit-image
RUN pip3 install import_ipynb

# Create a jupyter user
RUN useradd -ms /bin/bash jupyter

# Activate the jupyter user
USER jupyter

# Set the container working directory to the user home folder
WORKDIR /home/jupyter

# Start the jupyter notebook
ENTRYPOINT ["jupyter", "notebook", "--allow-root", "--ip=0.0.0.0", "--no-browser"]
```

Tabla 18. Contenido del fichero Dockerfile.

12.2. Bash Script: Creación del entorno con VirtualEnv

El script para generar el entorno de desarrollo bajo VirtualEnv se puede consultar en la Tabla 19.

```
#!/bin/bash
```

```
# Author:   Antonio González Hidalgo (agonzalezhidalgo@uoc.edu)
# TFM:      Traffic signals recognition
# Date:     Jan-2019

python3 --version
pip3 --version
virtualenv --version

echo Write the environment name
read envName

echo Do you want to install $envName environment?
read qvar

echo ========================================
if [ "$qvar" == "y" ]; then
  echo Creating $envName environment
  virtualenv --system-site-packages -p python3 ~/$envName
  echo Activating $envName environment
  source ~/$envName/bin/activate
  echo Updating pip
  pip3 install --upgrade pip

  echo Installing basic components
  pip3 install numpy scipy matplotlib ipython pandas sympy nose

  echo Installing Jupyter
  pip3 install jupyter

  echo Installing tensorflow
  pip3 install --upgrade tensorflow

  echo Installing Theano
  pip3 install theano

  echo Installing Keras
  pip3 install keras

  echo Installing scikit-image library
  pip install -U scikit-image

  echo Installing import_ipynb
  pip install import_ipynb

  echo Do you want to print software versions?
  read qvar
  if [ "$qvar" == "y" ]; then
    echo Jupyter version
    jupyter --version
    echo ========================================
    echo tensorflow: version:
```

```
    python -c "import tensorflow as tf; print(tf.__version__)"
    echo ==========================================
    echo Theano version
    python -c 'import theano; print(theano.__version__)'
    echo ==========================================
    echo Keras version
    python -c 'import keras; print(keras.__version__)'
  else
    echo [Skip] Show versions
  fi
else
  echo [Skip] Create environment
fi
```

Tabla 19. Contenido del fichero pythonEnv.sh

12.3. Jupyter Notebook: tfm_generic_functions

Conforme se ha ido creando redes neuronales se ha detectado mucho código común que no tenían nada que ver propiamente con la implementación de redes neuronales. Código referente al cargar imágenes, normalizarlas, recorrer directorios y ficheros, etc. Así que se ha creado un notebook de Jupyter encapsulando estas funciones básicas.

La interfaz pública del notebook queda definida por la Tabla 20.

```
# Devuelve una imagen
# - path: ruta de la imagen.
# - size: dimensión con la que se cargará la imagen.
# - as_gray: True para cargar la imagen en escala de grises.
def load_image(path, size, as_gray)


# Devuelve una lista con los datos del fichero csv.
# - path: ruta hasta el fichero csv.
# - delimeter: Carácter delimitador de campos
def read_csv(path, delimiter)


# Devuelve una colección con las imágenes y los labels de la ruta
# -datadir: path donde se encuentran la colección de imágenes.
# -shape: Dimensiones con las que se cargarán las imágenes.
# -as_gray: Indica si la imagen se cargará en escala de grises.
# -get_label_from_dir: (defecto True) específica si la categoría se lee del propio directorio
def readDataset(data_dir, shape, as_gray, get_label_from_dir = True)


# Imprime los tamaños de las colecciones
# - images: lista de imágenes precargadas.
# - labels: relación de las imágenes con las categorías a las que pertenecen.
# - np_images: np.array con las imágenes precargadas.
# - np_labels: np.array con las categorías precargadas.
```

```
# - environment: string identificativo del entorno.
def print_size_dataset(images, labels, np_images, np_labels, environment)


# Imprime una matriz 32x32 con los diferentes tipos de señales que
# se van a clasificar. Muestra la primera imagen de cada categoría.
# - images: lista de imágenes precargadas.
# - labels: relación de las imágenes con las categorías a las que pertenecen.
# - titles: lista con los nombres de las categorías.
def print_summary_dataset(images, labels, titles)


# Imprime todas las imágenes de una label especifica
# - label: índice de la label que se quiere buscar.
# - images: lista de imágenes precargadas.
# - source: lista de la relación de etiquetas.
# - titles: lista con las diferentes categorías de labels.
def print_signals(label, images, source, titles)


# Imprime los atributos de las imágenes de una label específica.
# - label: índice de la label que se quiere buscar.
# - images: lista de imágenes precargadas.
# - source: lista de la relación de etiquetas.
# - titles: lista con las diferentes categorías de labels.
def print_signals_attributes(label, images, source, titles)


# Devuelve una lista con las categorías de las imágenes de prueba leídas del fichero csv.
# - csv: fichero csv
# - class_column: número de columna que contiene las clases.
# - first_is_header: Indica si la primera fila es una cabecerá.
def get_class_id_array(csv, class_column, first_is_header = True)


# Devuelve un NP-Array con las images cargadas a un tamaño específico.
# - images: Lista de imagenes cargadas
# - size: tamaño al que se van a cargar las imágenes.
def get_array_images_test(images, size)
```

Tabla 20. Interfaz pública de tfm_generic_functions.

12.4. Jupyter notebook: Red densamente conectada – BelgiumTS

Se trata de la primera aproximación realizada con Keras para este trabajo. Utiliza el dataset BelgiumTS y entrena un modelo de red densamente conectada.

Reconocimiento de señales de tráfico

Máster Universitario en Ingeniería computacional y matemática - Área de inteligencia artificial

Antonio González Hidalgo (agonzalezhidalgo@uoc.edu)

Usando una red neuronal densamente conectada.

Para la correcta funcionamiento de este notebook, el notebook debe de estar estructurado de la siguiente manera:

- ./dataset
- ./dataset/labels.csv Fichero que contiene los nombres de las señales correspondientes.
- ./dataset/train Conjunto de imágenes que formarán el entrenamiento de la red neuronal.
- ./dataset/test Conjunto de imágenes que constituirán el test.

Este notebook se ha ejecutado usando el dataset de *BelgiumTS* https://btsd.ethz.ch/shareddata/

```
import import_ipynb
import tfm_generic_functions as traffic

# https://docs.python.org/3/library/os.html
import os

# https://docs.scipy.org/doc/numpy/reference/
import numpy as np

# https://keras.io/models/model/
import keras

# https://keras.io/models/sequential/
from keras.models import Sequential
```

```
importing Jupyter notebook from tfm_generic_functions.ipynb
```

```
Using TensorFlow backend.
```

```
# Obtenemos el directorio actual como trabajo.
ROOT_PATH = os.getcwd()

# Establecemos la dimensión de las imágenes.
IMG_SHAPE = (64, 64)
print("Tamaño de las imágenes de entrada: ", IMG_SHAPE)
IMG_SHAPE_LEN = IMG_SHAPE[0] * IMG_SHAPE[1]
print("Vectorizando la entrada, sería de un tamaño: ", IMG_SHAPE_LEN)

# Obtenemos los paths de trabajo
labels_path = os.path.join(ROOT_PATH, "dataset/labels.csv")
train_path = os.path.join(ROOT_PATH, "dataset/train")
test_path = os.path.join(ROOT_PATH, "dataset/test")
```

```
Tamaño de las imágenes de entrada:  (64, 64)
Vectorizando la entrada, sería de un tamaño:  4096
```

```
# Cargamos las imágenes de entrenamiento y de test.
images_train, labels_train = traffic.readDataset(train_path, IMG_SHAPE, True)
images_test, labels_test = traffic.readDataset(test_path, IMG_SHAPE, True)

# Convertimos las listas a array numpy de float32
np_images_train = np.asarray(images_train, dtype = np.float32)
np_labels_train = np.asarray(labels_train, dtype = np.int8)

np_images_test = np.asarray(images_test, dtype = np.float32)
np_labels_test = np.asarray(labels_test, dtype = np.int8)

# Recuperamos los nombres de las categorias. Los diferentes tipo de señales
# que se van a clasificar.
titles = traffic.read_csv(labels_path, ",")

# Se imprime información de los datos cargados.
traffic.print_size_dataset(images_train, labels_train, np_images_train, np_labels_train, "train")
traffic.print_size_dataset(images_test, labels_test, np_images_test, np_labels_test, "test")
print("Titles total: ", len(titles))
```

```
Total images (train):  4575
Total labels (train):  62
Images shape:  (4575, 64, 64)
Labels shape:  (4575,)
Total images (test):  2520
Total labels (test):  53
Images shape:  (2520, 64, 64)
Labels shape:  (2520,)
Titles total:  62
```

```
traffic.print_signals_attributes(10, images_train, labels_train, titles)
```

```
Signal:  A31
shape:  (64, 64)        min: 0.07119056843348817        max:  0.8444138641775234
shape:  (64, 64)        min: 0.0757897706346537         max:  0.9999984214974407
shape:  (64, 64)        min: 0.0810162723386273         max:  0.7385583086709786
shape:  (64, 64)        min: 0.0746757549329036         max:  0.779413426667588
shape:  (64, 64)        min: 0.03908460533994093        max:  0.9999536302239533
shape:  (64, 64)        min: 0.09013919558057597        max:  0.6983532050398296
shape:  (64, 64)        min: 0.04797929041398517        max:  0.7575448967657323
shape:  (64, 64)        min: 0.11363563903494187        max:  0.9999951098469267
shape:  (64, 64)        min: 0.07665419268439777        max:  0.7317439844228713
shape:  (64, 64)        min: 0.04622254174671689        max:  0.9999624135018614
shape:  (64, 64)        min: 0.09026138437603998        max:  0.7267406380456701
shape:  (64, 64)        min: 0.10657425202567813        max:  0.7553177768984263
shape:  (64, 64)        min: 0.05630743684531164        max:  0.9994308198688797
shape:  (64, 64)        min: 0.0001100447015161935      max:  0.6733553852206463
shape:  (64, 64)        min: 0.049955637898948475       max:  0.99712586963022
shape:  (64, 64)        min: 0.0673186777062532         max:  0.5417320245249639
shape:  (64, 64)        min: 0.08921070708265882        max:  0.8974986252857077
shape:  (64, 64)        min: 0.043456548359336614       max:  0.749569326440115
shape:  (64, 64)        min: 0.06768266319130192        max:  0.9315041474702751
shape:  (64, 64)        min: 0.06760005062327668        max:  0.9139797980574058
shape:  (64, 64)        min: 0.0846569978937958         max:  1.0
```

```
# Convertimos las imágenes en dos dimensiones
np_images_train = np_images_train.reshape(len(np_images_train), IMG_SHAPE_LEN)
np_images_test = np_images_test.reshape(len(np_images_test), IMG_SHAPE_LEN)
print("Nueva dimensión de imágenes: ", np_images_train.shape)

# Convertimos las labels de manera categórica
labels_categorical_train = keras.utils.to_categorical(np_labels_train)
labels_categorical_test = keras.utils.to_categorical(np_labels_test)

print("Ejemplo primera imagen de manera categórica: ", labels_categorical_train[0])
```

```
Nueva dimensión de imágenes:  (4575, 4096)
Ejemplo primera imagen de manera categórica:  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0
.
 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
# IMPLEMENTACIÓN RED NEURONAL
# En Keras la envoltura para cualquier red neuronal se crea con la clase Sequential
model = Sequential()

# Se deben de agregar capas a la red neuronal. Este ejemplo tendrá dos:
from keras.layers.core import Dense, Activation

# CAPA 1: Función de activación sigmoid
model.add(Dense(len(set(labels_train)), activation='selu', input_shape=(IMG_SHAPE_LEN,)))

# CAPA 2: Función de acivación softmax -> Devuelve una matrix de 10 valores con una probabilidad
# en cada posición.
model.add(Dense(len(set(labels_train)), activation='softmax'))

# Muestra la arquitectura de nuestra red neuronal
model.summary()

# Configurando el modelo de aprendizaje:
#  · loss, función para evaluar el grado de error entre salidas calculadas
#  · optimizador, función para calcular los pesos de los parámetros a partir de los datos de entrada
#  · metricas, para monitorizar el proceso de aprendizaje de la red.
model.compile(loss="categorical_crossentropy",
              optimizer="rmsprop",
              metrics=['accuracy'])

# Entrenamiento del modelo
# - batch_size, indica el número de datos que se usan en cada actualización.
# - epochs, indica el número de veces que se usan todos los datos del proceso.
model.fit(np_images_train, labels_categorical_train, batch_size=32, epochs=30)
```

```
# Evaluación del modelo
test_loss, test_acc = model.evaluate(np_images_test, labels_categorical_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)

y_pred = model.predict_classes(np_images_test)
acc = np.sum(y_pred == np_labels_test) / np.size(np_labels_test)
print("Test accuracy: {}".format(acc))
```

```
2520/2520 [==============================] - 0s 67us/step
Test loss: 0.6798687352769314
Test accuracy: 0.8742063493955703
Test accuracy: 0.8742063492063492
```

Esta red densamente conectada, con el dataset BelgiumTS, obtiene un porcentaje de precisión de 87,4206 %

12.5. Jupyter notebook: Red densamente conectada – German

Se trata de una red densamente conectada utilizando el dataset alemán.

Reconocimiento de señales de tráfico

Máster Universitario en Ingeniería computacional y matemática - Área de inteligencia artificial

Antonio González Hidalgo (agonzalezhidalgo@uoc.edu)

Usando una red neuronal densamente conectada.

Para la correcta funcionamiento de este notebook, el notebook debe de estar estructurado de la siguiente manera:

- ./dataset
- ./dataset/info.csv Fichero que contiene los nombres de las señales correspondientes.
- ./dataset/train/Images Conjunto de imágenes que formarán el entrenamiento de la red neuronal.
- ./dataset/test/info.csv. Fichero que contiene las categórias de las imágenes de test.
- ./dataset/test/Images Conjunto de imágenes que constituirán el test.

Para este notebook se ha usado el dataset *The German Traffic Sign Recognition Benchmark(GTSRB)* http://benchmark.ini.rub.de/?section=gtsrb&subsection=news

```
import import_ipynb
import tfm_generic_functions as traffic

# https://docs.python.org/3/library/os.html
import os

# https://docs.scipy.org/doc/numpy/reference/
import numpy as np

# https://keras.io/models/model/
import keras

# https://keras.io/models/sequential/
from keras.models import Sequential
```

```
Using TensorFlow backend.
```

```
# Obtenemos el directorio actual como trabajo.
ROOT_PATH = os.getcwd()

# Establecemos la dimensión de las imágenes.
IMG_SHAPE = (32, 32)
print("Tamaño de las imágenes de entrada: ", IMG_SHAPE)
IMG_SHAPE_LEN = IMG_SHAPE[0] * IMG_SHAPE[1]
print("Vectorizando la entrada, sería de un tamaño: ", IMG_SHAPE_LEN)

# Obtenemos los paths de trabajo
labels_path = os.path.join(ROOT_PATH, "dataset_ger/labels.csv")
train_path = os.path.join(ROOT_PATH, "dataset_ger/train/Images")
test_info_path = os.path.join(ROOT_PATH, "dataset_ger/test/GT-final_test.csv")
test_path = os.path.join(ROOT_PATH, "dataset_ger/test/Images")
```

```
Tamaño de las imágenes de entrada:  (32, 32)
Vectorizando la entrada, sería de un tamaño:  1024
```

```
# Cargamos las imágenes de entrenamiento.
images_train, labels_train = traffic.readDataset(train_path, IMG_SHAPE, True)

# Convertimos las listas a array numpy de float32
np_images_train = np.asarray(images_train, dtype = np.float32)
np_labels_train = np.asarray(labels_train, dtype = np.int8)

# Recuperamos los nombres de las categorias. Los diferentes tipo de señales
# que se van a clasificar.
titles = traffic.read_csv(labels_path, ",")

# Se imprime información de los datos cargados.
traffic.print_size_dataset(images_train, labels_train, np_images_train, np_labels_train, "train")
print("Titles total: ", len(titles))
```

```
Total images (train):  39209
Total labels (train):  43
Images shape:  (39209, 64, 64)
Labels shape:  (39209,)
Titles total:  43
```

```
# Cargamos las imágenes de entrenamiento.
images_test, labels_test = traffic.readDataset(test_path, IMG_SHAPE, True, False)

test_info  = traffic.read_csv(test_info_path, ";")
labels_test = traffic.get_class_id_array(test_info, 7)

# Convertimos las listas a array numpy de float32
np_images_test = np.asarray(images_test, dtype = np.float32)
np_labels_test = np.asarray(labels_test, dtype = np.int8)

# Se imprime información de los datos cargados.
traffic.print_size_dataset(images_test, labels_test, np_images_test, np_labels_test, "test")
traffic.print_signals_attributes(10, images_train, labels_train, titles)
```

```
Total images (test):  12630
Total labels (test):  43
Images shape:  (12630, 64, 64)
Labels shape:  (12630,)
Signal:  no overtaking (trucks) (prohibitory)
shape:  (64, 64)        min: 0.020456078431372548       max:  0.08046339690563724
shape:  (64, 64)        min: 0.07984509803921568        max:  0.5047765527343748
shape:  (64, 64)        min: 0.08628941176470588        max:  0.36627126247032055
shape:  (64, 64)        min: 0.04465786496629903        max:  0.41717529564950984
shape:  (64, 64)        min: 0.08632431372549021        max:  1.0
shape:  (64, 64)        min: 0.02017333333333333        max:  0.8124778980928266
shape:  (64, 64)        min: 0.06387607843137255        max:  0.8139148889399508
shape:  (64, 64)        min: 0.03585960784313726        max:  0.15088083807253375
shape:  (64, 64)        min: 0.1173756862745098         max:  0.5810357228477325
shape:  (64, 64)        min: 0.07535803921568628        max:  0.6346087365483302
shape:  (64, 64)        min: 0.11041843137254902        max:  0.47898996357038903
shape:  (64, 64)        min: 0.07764352941176471        max:  0.5751280937882997
shape:  (64, 64)        min: 0.05603294117647059        max:  0.9020766246361834
shape:  (64, 64)        min: 0.03725843137254902        max:  0.6073557441023284
shape:  (64, 64)        min: 0.10984588235294117        max:  0.5052557723699832
shape:  (64, 64)        min: 0.03445926549574915        max:  0.19286159737381275
```

```
# Convertimos las imágenes en dos dimensiones
np_images_train = np_images_train.reshape(len(np_images_train), IMG_SHAPE_LEN)
np_images_test = np_images_test.reshape(len(np_images_test), IMG_SHAPE_LEN)
print("Nueva dimensión de imágenes: ", np_images_train.shape)

# Convertimos las labels de manera categórica
labels_categorical_train = keras.utils.to_categorical(np_labels_train)
labels_categorical_test = keras.utils.to_categorical(np_labels_test)

print("Ejemplo primera imagen de manera categórica: ", labels_categorical_train[0])
```

```
Nueva dimensión de imágenes:  (39209, 4096)
Ejemplo primera imagen de manera categórica:  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0
.
 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
# IMPLEMENTACIÓN RED NEURONAL
# En Keras la envoltura para cualquier red neuronal se crea con la clase Sequential
model = Sequential()

# Se deben de agregar capas a la red neuronal. Este ejemplo tendrá dos:
from keras.layers.core import Dense, Activation

# CAPA 1: Función de activación sigmoid
model.add(Dense(len(set(labels_train)), activation='selu', input_shape=(IMG_SHAPE_LEN,)))

# CAPA 2: Función de acivación softmax -> Devuelve una matrix de 10 valores con una probabilidad
# en cada posición.
model.add(Dense(len(set(labels_train)), activation='softmax'))

# Muestra la arquitectura de nuestra red neuronal
model.summary()

# Configurando el modelo de aprendizaje:
#  · loss, función para evaluar el grado de error entre salidas calculadas
#  · optimizador, función para calcular los pesos de los parámetros a partir de los datos de entrada
#  · metricas, para monitorizar el proceso de aprendizaje de la red.
model.compile(loss="categorical_crossentropy",
             optimizer="rmsprop",
             metrics=['accuracy'])

# Entrenamiento del modelo
# - batch_size, indica el número de datos que se usan en cada actualización.
# - epochs, indica el número de veces que se usan todos los datos del proceso.
model.fit(np_images_train, labels_categorical_train, batch_size=32, epochs=30)
```

```
# Evaluación del modelo
test_loss, test_acc = model.evaluate(np_images_test, labels_categorical_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)

y_pred = model.predict_classes(np_images_test)
acc = np.sum(y_pred == np_labels_test) / np.size(np_labels_test)
print("Test accuracy: {}".format(acc))
```

```
12630/12630 [==============================] - 0s 25us/step
Test loss: 13.406682038326068
Test accuracy: 0.037608867775138856
Test accuracy: 0.037608867775138856
```

Vemos como se ha obtenido un porcentaje de precisión bastante bajo: 3,760886 %.

12.6. Jupyter notebook: Red convolucional – BelgiumTS

Se entrenará una CNN usando el dataset *BelgiumTS.*

Reconocimiento de señales de tráfico ¶

Máster Universitario en Ingeniería computacional y matemática - Área de inteligencia artificial

Antonio González Hidalgo (agonzalezhidalgo@uoc.edu)

Usando una red neuronal convolucional.

Para la correcta funcionamiento de este notebook, el notebook debe de estar estructurado de la siguiente manera:

- ./dataset
- ./dataset/info.csv Fichero que contiene los nombres de las señales correspondientes.
- ./dataset/train/Images Conjunto de imágenes que formarán el entrenamiento de la red neuronal.
- ./dataset/test/info.csv. Fichero que contiene las categórias de las imágenes de test.
- ./dataset/test/Images Conjunto de imágenes que constituirán el test.

Este notebook se ha ejecutado usando el dataset de *BelgiumTS* https://btsd.ethz.ch/shareddata/

```
import import_ipynb
import tfm_generic_functions as traffic

# https://docs.python.org/3/library/os.html
import os

# https://docs.scipy.org/doc/numpy/reference/
import numpy as np

# https://keras.io/models/model/
import keras

from keras import models

# Core Layers: https://keras.io/layers/core/
# Convolution Layers: https://keras.io/layers/convolutional/
from keras import layers

# https://keras.io/preprocessing/image/
from keras.preprocessing.image import ImageDataGenerator

# https://keras.io/callbacks/
from keras.callbacks import ModelCheckpoint
```

```
importing Jupyter notebook from tfm_generic_functions.ipynb
```

```
Using TensorFlow backend.
```

```
# Obtenemos el directorio actual como trabajo.
ROOT_PATH = os.getcwd()

# Establecemos la dimensión de las imágenes.
IMG_SHAPE = (64, 64)

# Configuramos la CNN
EPOCHS = 25
BATCH_SIZE = 128
ACTIVATION = 'selu'
OPTIMIZER = 'rmsprop'

print("Tamaño de las imágenes de entrada: ", IMG_SHAPE)
IMG_SHAPE_LEN = IMG_SHAPE[0] * IMG_SHAPE[1]
print("Vectorizando la entrada, sería de un tamaño: ", IMG_SHAPE_LEN)

# Obtenemos los paths de trabajo
labels_path = os.path.join(ROOT_PATH, "dataset_bel/labels.csv")
train_path = os.path.join(ROOT_PATH, "dataset_bel/train")
test_path = os.path.join(ROOT_PATH, "dataset_bel/test")
```

```
Tamaño de las imágenes de entrada:  (64, 64)
Vectorizando la entrada, sería de un tamaño:  4096
```

```
# Cargamos las imágenes de entrenamiento y de test.
images_train, labels_train = traffic.readDataset(train_path, IMG_SHAPE, False)
images_test, labels_test = traffic.readDataset(test_path, IMG_SHAPE, False)

# Convertimos las listas a array numpy de float32
np_images_train = np.asarray(images_train, dtype = np.float32)
np_labels_train = np.asarray(labels_train, dtype = np.int8)

np_images_test = np.asarray(images_test, dtype = np.float32)
np_labels_test = np.asarray(labels_test, dtype = np.int8)

# Recuperamos los nombres de las categorias. Los diferentes tipo de señales
# que se van a clasificar.
titles = traffic.read_csv(labels_path, ",")

# Se imprime información de los datos cargados.
traffic.print_size_dataset(images_train, labels_train, np_images_train, np_labels_train, "train")
traffic.print_size_dataset(images_test, labels_test, np_images_test, np_labels_test, "test")
print("Titles total: ", len(titles))
traffic.print_signals_attributes(10, images_train, labels_train, titles)
```

```
Total images (train):  4575
Total labels (train):  62
Images shape:  (4575, 64, 64, 3)
Labels shape:  (4575,)
Total images (test):  2520
Total labels (test):  53
Images shape:  (2520, 64, 64, 3)
Labels shape:  (2520,)
Titles total:  62
Signal:  A31
shape:  (64, 64, 3)     min: 0.04844037224264742        max:  0.9921568627450981
shape:  (64, 64, 3)     min: 0.012683823529412207       max:  1.0
```

```
# Convertimos las labels de manera categórica
labels_categorical_train = keras.utils.to_categorical(np_labels_train)
labels_categorical_test = keras.utils.to_categorical(np_labels_test)

print("Ejemplo primera imagen de manera categórica: ", labels_categorical_train[0])
```

```
Ejemplo primera imagen de manera categórica:  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0
.
 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
def get_keras_model(activation):
    # IMPLEMENTACIÓN RED NEURONAL
    # En Keras la envoltura para cualquier red neuronal se crea con la clase Sequential
    model = models.Sequential()

    model.add(layers.Conv2D(32, (5, 5),
                            activation=activation, input_shape=(IMG_SHAPE[0], IMG_SHAPE[1], 3)))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(64, (5, 5), activation=activation))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(128, (5, 5), activation=activation))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Flatten())
    model.add(layers.Dense(len(set(labels_train)), activation='softmax'))
    return model
```

```
model = get_keras_model(ACTIVATION)

# Muestra la arquitectura de nuestra red neuronal
model.summary()

# Configurando el modelo de aprendijaze:
#  · loss, función para evaluar el grado de error entre salidas calculadas
#  · optimizador, función para calcular los pesos de los parámetros a partir de los datos de entrada
#  · metricas, para monitorizar el proceso de aprendizaje de la red.
model.compile(loss="categorical_crossentropy",
              optimizer=OPTIMIZER,
              metrics=['accuracy'])

# Entrenamiento del modelo
# - batch_size, indica el número de datos que se usan en cada actualización.
# - epochs, indica el número de veces que se usan todos los datos del proceso.
#model.fit(np_images_train, labels_categorical_train, batch_size=32, epochs=20)
model.fit(np_images_train, labels_categorical_train,
          batch_size = BATCH_SIZE,
          epochs = EPOCHS,
          callbacks=[ModelCheckpoint('model_64_64_bel.h5', save_best_only = False)])

# Evaluación del modelo
test_loss, test_acc = model.evaluate(np_images_test, labels_categorical_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
# Evaluación del modelo
test_loss, test_acc = model.evaluate(np_images_test, labels_categorical_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
2520/2520 [==============================] - 7s 3ms/step
Test loss: 0.14018301860972385
Test accuracy: 0.9781746031746031
```

Con esta red se obtiene un porcentaje de precisión de 97,8174%. El fichero de pesos generado es el ‘model_64_64_bel.h5’.

Utilizando esta misma red, se introduce el concepto de ‘ImageDataGenerator’, estudiado en el punto 4.5.

```
datagen = ImageDataGenerator(featurewise_center=False,
                             featurewise_std_normalization=False,
                             width_shift_range=0.1,
                             height_shift_range=0.1,
                             zoom_range=0.2,
                             shear_range=0.1,
                             rotation_range=10.,)

datagen.fit(np_images_test)
```

```
model_2 = get_keras_model(ACTIVATION)

model_2.compile(loss="categorical_crossentropy",
              optimizer=OPTIMIZER,
              metrics=['accuracy'])

model_2.fit_generator(datagen.flow(np_images_train, labels_categorical_train, batch_size=32),
                      steps_per_epoch = np_images_train.shape[0],
                      epochs = EPOCHS,
                      callbacks=[ModelCheckpoint('model_data_aug_64_64_bel.h5',save_best_only = False)])
```

```
# Evaluación del modelo
test_loss, test_acc = model_2.evaluate(np_images_test, labels_categorical_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
2520/2520 [==============================] - 8s 3ms/step
Test loss: 0.2915069095672123
Test accuracy: 0.9817460317460317
```

Usando los datos generados con ImageDataGenerator se ha obtenido unos resultados mejores alcanzando los 98,1746% de precisión. El fichero de pesos generado es el 'model_data_aug_64_64_bel.h5'.

12.7. Jupyter notebook: Red convolucional – German

Se entrenará una CNN usando el dataset *German*.

Reconocimiento de señales de tráfico

Máster Universitario en Ingeniería computacional y matemática - Área de inteligencia artificial

Antonio González Hidalgo (agonzalezhidalgo@uoc.edu) ¶

Usando una red neuronal convolucional.

Para la correcta funcionamiento de este notebook, el notebook debe de estar estructurado de la siguiente manera:

- ./dataset
- ./dataset/info.csv Fichero que contiene los nombres de las señales correspondientes.
- ./dataset/train/Images Conjunto de imágenes que formarán el entrenamiento de la red neuronal.
- ./dataset/test/info.csv. Fichero que contiene las categórias de las imágenes de test.
- ./dataset/test/Images Conjunto de imágenes que constituirán el test.

Para este notebook se ha usado el dataset *The German Traffic Sign Recognition Benchmark(GTSRB)* http://benchmark.ini.rub.de/?section=gtsrb&subsection=news

```
import import_ipynb
import tfm_generic_functions as traffic

# https://docs.python.org/3/library/os.html
import os

# https://docs.scipy.org/doc/numpy/reference/
import numpy as np

# https://keras.io/models/model/
import keras

from keras import models

# Core Layers: https://keras.io/layers/core/
# Convolution Layers: https://keras.io/layers/convolutional/
from keras import layers

# https://keras.io/preprocessing/image/
from keras.preprocessing.image import ImageDataGenerator

# https://keras.io/callbacks/
from keras.callbacks import ModelCheckpoint
```

```
importing Jupyter notebook from tfm_generic_functions.ipynb
```

```
Using TensorFlow backend.
```

```
# Obtenemos el directorio actual como trabajo.
ROOT_PATH = os.getcwd()

# Establecemos la dimensión de las imágenes.
IMG_SHAPE = (32, 32)

# Configurando la CNN
EPOCHS = 25
BATCH_SIZES = 128
ACTIVATION = 'selu'
OPTIMIZER = 'rmsprop'

print("Tamaño de las imágenes de entrada: ", IMG_SHAPE)
IMG_SHAPE_LEN = IMG_SHAPE[0] * IMG_SHAPE[1]
print("Vectorizando la entrada, sería de un tamaño: ", IMG_SHAPE_LEN)

# Obtenemos los paths de trabajo
labels_path = os.path.join(ROOT_PATH, "dataset_ger/labels.csv")
train_path = os.path.join(ROOT_PATH, "dataset_ger/train/Images")
test_info_path = os.path.join(ROOT_PATH, "dataset_ger/test/GT-final_test.csv")
test_path = os.path.join(ROOT_PATH, "dataset_ger/test/Images")
```

```
Tamaño de las imágenes de entrada:  (32, 32)
Vectorizando la entrada, sería de un tamaño:  1024
```

```
# Cargamos las imágenes de entrenamiento.
images_train, labels_train = traffic.readDataset(train_path, IMG_SHAPE, False)

# Convertimos las listas a array numpy de float32
np_images_train = np.asarray(images_train, dtype = np.float32)
np_labels_train = np.asarray(labels_train, dtype = np.int8)

# Recuperamos los nombres de las categorias. Los diferentes tipo de señales
# que se van a clasificar.
titles = traffic.read_csv(labels_path, ",")

# Se imprime información de los datos cargados.
traffic.print_size_dataset(images_train, labels_train, np_images_train, np_labels_train, "train")
print("Titles total: ", len(titles))
```

```
Total images (train):  39209
Total labels (train):  43
Images shape:  (39209, 32, 32, 3)
Labels shape:  (39209,)
Titles total:  43
```

```
# Cargamos las imágenes de entrenamiento.
images_test, labels_test = traffic.readDataset(test_path, IMG_SHAPE, False, False)

test_info  = traffic.read_csv(test_info_path, ";")
labels_test = traffic.get_class_id_array(test_info, 7)

# Convertimos las listas a array numpy de float32
np_images_test = np.asarray(images_test, dtype = np.float32)
np_labels_test = np.asarray(labels_test, dtype = np.int8)

# Se imprime información de los datos cargados.
traffic.print_size_dataset(images_test, labels_test, np_images_test, np_labels_test, "test")
```

```
Total images (test):  12630
Total labels (test):  43
Images shape:  (12630, 32, 32, 3)
Labels shape:  (12630,)
```

```
# Convertimos las labels de manera categórica
labels_categorical_train = keras.utils.to_categorical(np_labels_train)
labels_categorical_test = keras.utils.to_categorical(np_labels_test)

print("Ejemplo primera imagen de manera categórica: ", labels_categorical_train[0])
```

```
Ejemplo primera imagen de manera categórica:  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0
.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
def get_keras_model(activation):
    # IMPLEMENTACIÓN RED NEURONAL
    # En Keras la envoltura para cualquier red neuronal se crea con la clase Sequential
    model = models.Sequential()

    model.add(layers.Conv2D(32, (5, 5),
                            activation=activation, input_shape=(IMG_SHAPE[0], IMG_SHAPE[1], 3)))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(64, (5, 5), activation=activation))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Flatten())
    model.add(layers.Dense(len(set(labels_train)), activation='softmax'))
    return model
```

```
model = get_keras_model(ACTIVATION)

# Muestra la arquitectura de nuestra red neuronal
model.summary()

# Configurando el modelo de aprendijaze:
#  · loss, función para evaluar el grado de error entre salidas calculadas
#  · optimizador, función para calcular los pesos de los parámetros a partir de los datos de entrada
#  · metricas, para monitorizar el proceso de aprendizaje de la red.
model.compile(loss="categorical_crossentropy",
              optimizer=OPTIMIZER,
              metrics=['accuracy'])

# Entrenamiento del modelo
# - batch_size, indica el número de datos que se usan en cada actualización.
# - epochs, indica el número de veces que se usan todos los datos del proceso.
#model.fit(np_images_train, labels_categorical_train, batch_size=32, epochs=20)
model.fit(np_images_train, labels_categorical_train,
          batch_size = BATCH_SIZES,
          verbose = 2,
          epochs = EPOCHS,
          callbacks=[ModelCheckpoint('model_32_32_ger.h5', save_best_only = False)])

# Evaluación del modelo
test_loss, test_acc = model.evaluate(np_images_test, labels_categorical_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
Epoch 25/25
 - 71s - loss: 0.0468 - acc: 0.9874
12630/12630 [==============================] - 7s 541us/step
Test loss: 15.099030353639773
Test accuracy: 0.03737133808392716
```

Con esta red se obtiene un porcentaje de precisión de 3,7371%. El fichero de pesos generado es el ‘model_32_32_ger.h5’

```
datagen = ImageDataGenerator(featurewise_center=False,
                             featurewise_std_normalization=False,
                             width_shift_range=0.1,
                             height_shift_range=0.1,
                             zoom_range=0.2,
                             shear_range=0.1,
                             rotation_range=10.,)

datagen.fit(np_images_test)
```

```
model_2 = get_keras_model(ACTIVATION)

model_2.compile(loss="categorical_crossentropy",
             optimizer=OPTIMIZER,
             metrics=['accuracy'])

model_2.fit_generator(datagen.flow(np_images_train, labels_categorical_train, batch_size=32),
                      steps_per_epoch = np_images_train.shape[0],
                      epochs = EPOCHS,
                      verbose = 2,
                      callbacks=[ModelCheckpoint('model_data_aug_32_32_ger.h5',save_best_only = False)])
```

```
# Evaluación del modelo
test_loss, test_acc = model_2.evaluate(np_images_test, labels_categorical_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
12630/12630 [==============================] - 5s 403us/step
Test loss: 15.23345929626803
Test accuracy: 0.037925574030087096
```

Después de entrenar otra red usando el generador de imágenes, podemos ver como obtenemos unos valores de precisión más elevados que con el conjunto de datos básicos, pero aún siguen siendo muy ineficientes, lejos de los alcanzados con el dataset *BelgiumTS*.

12.8. Jupyter notebook: Red convolucional – Detector

Se entrenará una red neuronal capaz de detectar si existe una señal de tráfico o no.

Reconocimiento de señales de tráfico

Máster Universitario en Ingeniería computacional y matemática - Área de inteligencia artificial

Antonio González Hidalgo (agonzalezhidalgo@uoc.edu)

Usando una red neuronal convolucional.

Para la correcta funcionamiento de este notebook, el notebook debe de estar estructurado de la siguiente manera:

- ./dataset
- ./dataset/info.csv Fichero que contiene los nombres de las señales correspondientes.
- ./dataset/train/Images Conjunto de imágenes que formarán el entrenamiento de la red neuronal.
- ./dataset/test/info.csv. Fichero que contiene las categórias de las imágenes de test.
- ./dataset/test/Images Conjunto de imágenes que constituirán el test.

```
import import_ipynb
import tfm_generic_functions as traffic

# https://docs.python.org/3/library/os.html
import os

# https://docs.scipy.org/doc/numpy/reference/
import numpy as np

# https://keras.io/models/model/
import keras

from keras import models

# Core Layers: https://keras.io/layers/core/
# Convolution Layers: https://keras.io/layers/convolutional/
from keras import layers

# https://keras.io/preprocessing/image/
from keras.preprocessing.image import ImageDataGenerator

# https://keras.io/callbacks/
from keras.callbacks import ModelCheckpoint
```

```
importing Jupyter notebook from tfm_generic_functions.ipynb
```

```
Using TensorFlow backend.
```

```
# Obtenemos el directorio actual como trabajo.
ROOT_PATH = os.getcwd()

# Establecemos la dimensión de las imágenes.
IMG_SHAPE = (64, 64)

# Configuración de la CNN
ACTIVATION = "elu"
OPTIMIZER = "adam"
EPOCH = 25
BATCH_SIZE = 64

print("Tamaño de las imágenes de entrada: ", IMG_SHAPE)
IMG_SHAPE_LEN = IMG_SHAPE[0] * IMG_SHAPE[1]
print("Vectorizando la entrada, sería de un tamaño: ", IMG_SHAPE_LEN)

# Obtenemos los paths de trabajo
train_path = os.path.join(ROOT_PATH, "dataset_det/train")
test_path = os.path.join(ROOT_PATH, "dataset_det/test")
```

```
Tamaño de las imágenes de entrada:  (64, 64)
Vectorizando la entrada, sería de un tamaño:  4096
```

```
# Cargamos las imágenes de entrenamiento y de test.
images_train, labels_train = traffic.readDataset(train_path, IMG_SHAPE, False)
images_test, labels_test = traffic.readDataset(test_path, IMG_SHAPE, False)

# Convertimos las listas a array numpy de float32
np_images_train = np.asarray(images_train, dtype = np.float32)
np_labels_train = np.asarray(labels_train, dtype = np.int8)

np_images_test = np.asarray(images_test, dtype = np.float32)
np_labels_test = np.asarray(labels_test, dtype = np.int8)


# Se imprime información de los datos cargados.
traffic.print_size_dataset(images_train, labels_train, np_images_train, np_labels_train, "train")
traffic.print_size_dataset(images_test, labels_test, np_images_test, np_labels_test, "test")
```

```
Total images (train):  15481
Total labels (train):  2
Images shape:  (15481, 64, 64, 3)
Labels shape:  (15481,)
Total images (test):  14726
Total labels (test):  2
Images shape:  (14726, 64, 64, 3)
Labels shape:  (14726,)
```

```
# Convertimos las labels de manera categórica
labels_categorical_train = keras.utils.to_categorical(np_labels_train)
labels_categorical_test = keras.utils.to_categorical(np_labels_test)

print("Ejemplo primera imagen de manera categórica: ", labels_categorical_train[0])
```

```
Ejemplo primera imagen de manera categórica:  [1. 0.]
```

```
def get_keras_model(activation):
    # IMPLEMENTACIÓN RED NEURONAL
    # En Keras la envoltura para cualquier red neuronal se crea con la clase Sequential
    model = models.Sequential()

    model.add(layers.Conv2D(32, (5, 5),
                            activation=activation, input_shape=(IMG_SHAPE[0], IMG_SHAPE[1], 3)))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(64, (5, 5), activation=activation))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(128, (5, 5), activation=activation))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Flatten())
    model.add(layers.Dense(len(set(labels_train)), activation='softmax'))
    return model
```

```
model = get_keras_model(ACTIVATION)

# Muestra la arquitectura de nuestra red neuronal
model.summary()

# Configurando el modelo de aprendijaze:
#  · loss, función para evaluar el grado de error entre salidas calculadas
#  · optimizador, función para calcular los pesos de los parámetros a partir de los datos de entrada
#  · metricas, para monitorizar el proceso de aprendizaje de la red.
model.compile(loss="categorical_crossentropy",
              optimizer=OPTIMIZER,
              metrics=['accuracy'])

# Entrenamiento del modelo
# - batch_size, indica el número de datos que se usan en cada actualización.
# - epochs, indica el número de veces que se usan todos los datos del proceso.
#model.fit(np_images_train, labels_categorical_train, batch_size=32, epochs=20)
model.fit(np_images_train, labels_categorical_train,
          batch_size = BATCH_SIZE,
          epochs = EPOCH,
          verbose = 2,
          callbacks=[ModelCheckpoint('model_64_64_det.h5', save_best_only = False)])

# Evaluación del modelo
test_loss, test_acc = model.evaluate(np_images_test, labels_categorical_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
Epoch 24/25
 - 89s - loss: 0.0358 - acc: 0.9959
Epoch 25/25
 - 88s - loss: 0.0323 - acc: 0.9955
14726/14726 [==============================] - 25s 2ms/step
Test loss: 1.980566535920864
Test accuracy: 0.8227624609534158
```

Con este primer modelo se obtiene un 82,27% de precisión. El fichero de pesos generado es el ‘model_64_64_det.h5’.

```
datagen = ImageDataGenerator(featurewise_center=False,
                             featurewise_std_normalization=False,
                             width_shift_range=0.1,
                             height_shift_range=0.1,
                             zoom_range=0.2,
                             shear_range=0.1,
                             rotation_range=10.,)

datagen.fit(np_images_test)
```

```
model_2 = get_keras_model(ACTIVATION)

model_2.compile(loss="categorical_crossentropy",
              optimizer=OPTIMIZER,
              metrics=['accuracy'])

model_2.fit_generator(datagen.flow(np_images_train, labels_categorical_train, batch_size=32),
                      steps_per_epoch = np_images_train.shape[0],
                      epochs = EPOCH,
                      callbacks=[ModelCheckpoint('model_data_aug_64_64_det.h5',save_best_only = False)])
```

```
# Evaluación del modelo
test_loss, test_acc = model_2.evaluate(np_images_test, labels_categorical_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
14726/14726 [==============================] - 22s 1ms/step
Test loss: 3.1829024738516156
Test accuracy: 0.8025261442346869
```

Nuevamente utilizando el ImageDataGenerator se ha obtenido un valor de precisión de 80,25% algo menor. El fichero de pesos generado es el ‘model_data_aug_64_64_det.h5’.

12.9. Jupyter notebook: Test detector

El objetivo de esta prueba es facilitar 3 imágenes completas y trocearlas en pequeñas celdas. Cada una de estas celdas se pasará al detector entrenando en el punto 0 y ver en cuantas celdas se ha detectado una imagen de tráfico.

```
import import_ipynb
import tfm_generic_functions as traffic

# https://docs.python.org/3/library/os.html
import os

# http://scikit-image.org/docs/stable/api/api.html
import skimage
from skimage import data, io

# https://docs.scipy.org/doc/numpy/reference/
import numpy as np

import keras

from keras import models

# Core Layers: https://keras.io/layers/core/
# Convolution Layers: https://keras.io/layers/convolutional/
from keras import layers

# https://matplotlib.org/api/index.html
import matplotlib

# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot
import matplotlib.pyplot as plt
```

```
importing Jupyter notebook from tfm_generic_functions.ipynb
```

```
Using TensorFlow backend.
```

```
OFFSET = 1
IMG_SHAPE = (64, 64)
ROOT_PATH = os.getcwd()
```

```
def get_keras_model_detector(classes):
    # IMPLEMENTACIÓN RED NEURONAL
    # En Keras la envoltura para cualquier red neuronal se crea con la clase Sequential
    model = models.Sequential()

    model.add(layers.Conv2D(32, (5, 5),
                            activation='elu', input_shape=(IMG_SHAPE[0], IMG_SHAPE[1], 3)))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(64, (5, 5), activation='elu'))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(128, (5, 5), activation='elu'))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Flatten())
    model.add(layers.Dense(classes, activation='softmax'))
    return model
```

```
def load_trained_model_detector(weights_path):
    model = get_keras_model_detector(2)
    model.load_weights(weights_path)
    return model
```

```
model_detector = load_trained_model_detector(os.path.join(ROOT_PATH, "model_data_aug_64_64_gpu_detector.h5"))
model_detector.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = ['accuracy'])
```

```
images_test_path = [os.path.join(ROOT_PATH, "images_test/Prohibido_001.jpg"),
                    os.path.join(ROOT_PATH, "images_test/Rotonda_001.jpg"),
                    os.path.join(ROOT_PATH, "images_test/Stop_001.jpg")]
```

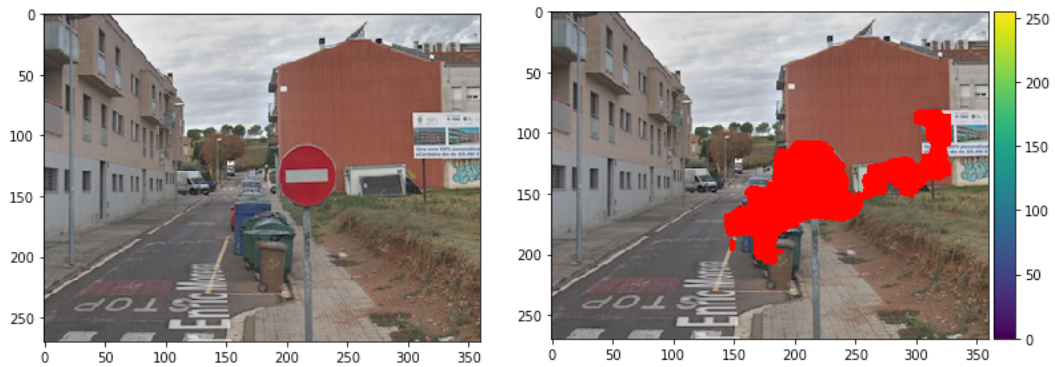
```
def get_tiles(image, tile_shape, offset):
    tiles = []
    xx = tile_shape[0]
    yy = tile_shape[1]
    for x in range(0, image.shape[0]- xx, offset):
        for y in range(0, image.shape[1] - yy, offset):
            tile = image[x:x + xx, y:y + yy]
            tiles.append(tile)
    return tiles
```

```
def get_normalized_image(image_path):
    image_test = skimage.data.imread(image_path)
    height = image_test.shape[0]
    width = image_test.shape[1]
    return skimage.transform.resize(image_test, (height, width), mode='constant')
```

```
def getCenter(index, img_size, img_shape):
    img_width = img_size[1]
    img_heigth = img_size[0]
    sh_width = img_shape[1]
    sh_height = img_shape[0]
    max_row = img_width - sh_width

    row = int(index / max_row)
    col = index % max_row

    centerY = row + (sh_height / 2)
    centerX = col + (sh_width / 2)

    center = (centerX, centerY)
    return center
```

```
def paintPointOnImage(x, y, img):
    color = [255, 0, 0]
    point_size = 3
    offset = int(point_size / 2)
    for xx in range(x - offset, x + offset + 1):
        for yy in range(y - offset, y + offset + 1):
            img[xx, yy] = color
```

```
for image_path in images_test_path:
    image_test = get_normalized_image(image_path)
    image_edit = get_normalized_image(image_path)
    io.imshow(image_test)
    plt.show()
    image_tiles = get_tiles(image_test, IMG_SHAPE, OFFSET)
    detected_tiles = []
    for index in range(0, len(image_tiles)):
        tile = image_tiles[index]
        np_tile = np.expand_dims(tile, axis=0)
        result = model_detector.predict_classes(np_tile)
        if result == 1:
            center = getCenter(index, image_test.shape, IMG_SHAPE)
            paintPointOnImage(int(center[1]), int(center[0]), image_edit)
            detected_tiles.append(tile)
    print("{} - Detected traffic signs in {} tiles".format(image_path, len(detected_tiles)))
    io.imshow(image_edit)
    plt.show()
```




0
50
100
150
200
250
300
350
STOP

12.10. Jupyter notebook: Test clasificación

Tiene por finalidad ejecutar una prueba (con imágenes recortadas) para testear los clasificadores de imágenes entrenados con el dataset *BelgiumTS* y *German*.

```
import import_ipynb
import tfm_generic_functions as traffic

# https://docs.python.org/3/library/os.html
import os

# http://scikit-image.org/docs/stable/api/api.html
import skimage
from skimage import data, io

# https://docs.scipy.org/doc/numpy/reference/
import numpy as np

import keras

from keras import models

# Core Layers: https://keras.io/layers/core/
# Convolution Layers: https://keras.io/layers/convolutional/
from keras import layers

# https://matplotlib.org/api/index.html
import matplotlib

# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot
import matplotlib.pyplot as plt
```

```
importing Jupyter notebook from tfm_generic_functions.ipynb
```

```
Using TensorFlow backend.
```

```
# Obtenemos el directorio actual como trabajo.
ROOT_PATH = os.getcwd()

weights_bel = "model_data_aug_64_64_bel_gpu.h5"
weights_ger = "model_data_aug_32_32_gpu_ger.h5"

# Obtenemos los paths de trabajo
weights_path_bel = os.path.join(ROOT_PATH, weights_bel)
weights_path_ger = os.path.join(ROOT_PATH, weights_ger)

labels_path_bel = os.path.join(ROOT_PATH, "dataset_bel/labels.csv")
labels_path_ger = os.path.join(ROOT_PATH, "dataset_ger/labels.csv")
```

```
# Recuperamos los nombres de las categorias. Los diferentes tipo de señales
# que se van a clasificar.
titles_bel = traffic.read_csv(labels_path_bel, ",")
titles_ger = traffic.read_csv(labels_path_ger, ",")
```

```
def get_keras_model_bel(size_categories, shape):
    # IMPLEMENTACIÓN RED NEURONAL
    # En Keras la envoltura para cualquier red neuronal se crea con la clase Sequential
    model = models.Sequential()

    model.add(layers.Conv2D(32, (5, 5),
                            activation = 'selu', input_shape = shape))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(64, (5, 5), activation='selu'))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(128, (5, 5), activation='selu'))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Flatten())
    model.add(layers.Dense(size_categories, activation='softmax'))
    return model

def load_trained_model_bel(weights_path, size_categories, shape):
    model = get_keras_model_bel(size_categories, shape)
    model.load_weights(weights_path)
    return model
```

```
def get_keras_model_ger(size_categories, shape):
    # IMPLEMENTACIÓN RED NEURONAL
    # En Keras la envoltura para cualquier red neuronal se crea con la clase Sequential
    model = models.Sequential()

    model.add(layers.Conv2D(32, (5, 5),
                            activation='selu', input_shape=shape))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(64, (5, 5), activation='selu'))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Flatten())
    model.add(layers.Dense(size_categories, activation='softmax'))
    return model

def load_trained_model_ger(weights_path, size_categories, shape):
    model = get_keras_model_ger(size_categories, shape)
    model.load_weights(weights_path)
    return model
```

```
model_bel = load_trained_model_bel(weights_path_bel, len(titles_bel), (64, 64, 3))
model_ger = load_trained_model_ger(weights_path_ger, len(titles_ger), (32, 32, 3))

model_bel.compile(loss="categorical_crossentropy", optimizer="rmsprop", metrics=['accuracy'])
model_ger.compile(loss="categorical_crossentropy", optimizer="rmsprop", metrics=['accuracy'])
```

```
WARNING:tensorflow:From /Users/togohi/traffic/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.
py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /Users/togohi/traffic/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:3445: c
alling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future versi
on.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

```
images_test_path = [os.path.join(ROOT_PATH, "images_test/Prohibido_001_recorte.jpg"),
                    os.path.join(ROOT_PATH, "images_test/Rotonda_001_recorte.jpg"),
                    os.path.join(ROOT_PATH, "images_test/Stop_001_recorte.jpg")]
```

```
images_test_bel = get_array_images_test((64, 64))
images_test_ger = get_array_images_test((32, 32))


for img in images_test_bel:
    print("Dimensiones de la imagen: {}".format(img.shape))
    print("Comprobamos que la imagen este normalizada: MIN: {} MAX: {}".format(img.min(), img.max()))
    plt.imshow(img)
    plt.show()
```

```
print("Belgium:")
class_result_bel = model_bel.predict_classes(images_test_bel)
for result_bel in class_result_bel:
    print("Clase resultante: {} y título: {}". format(result_bel, titles_bel[result_bel]))


print("Germany:")
class_result_ger = model_ger.predict_classes(images_test_ger)
for result_ger in class_result_ger:
    print("Clase resultante: {} y título: {}". format(result_ger, titles_ger[result_ger]))
```

```
Belgium:
Clase resultante: 22 y título: ['22', 'C1']
Clase resultante: 37 y título: ['37', 'D5']
Clase resultante: 21 y título: ['21', 'B5']
Germany:
Clase resultante: 17 y título: ['17', 'no entry (other)']
Clase resultante: 40 y título: ['40', 'roundabout (mandatory)']
Clase resultante: 14 y título: ['14', 'stop (other)']
```

12.11. Jupyter notebook: Test clasificación (Imagen completa)

Tiene por finalidad ejecutar una prueba (con imágenes recortadas) para testear los clasificadores de imágenes entrenados con el dataset *BelgiumTS* y *German*.

```
import import_ipynb
import tfm_generic_functions as traffic

import os

# https://keras.io/models/model/
import keras

# https://docs.scipy.org/doc/numpy/reference/
import numpy as np

from keras import models

# Core Layers: https://keras.io/layers/core/
# Convolution Layers: https://keras.io/layers/convolutional/
from keras import layers

# https://matplotlib.org/api/index.html
import matplotlib

# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot
import matplotlib.pyplot as plt
```

```
importing Jupyter notebook from tfm_generic_functions.ipynb
```

```
Using TensorFlow backend.
```

```
# Obtenemos el directorio actual como trabajo.
ROOT_PATH = os.getcwd()

weights_bel = "model_data_aug_64_64_bel_gpu.h5"
weights_ger = "model_data_aug_32_32_gpu_ger.h5"

# Obtenemos los paths de trabajo
weights_path_bel = os.path.join(ROOT_PATH, weights_bel)
weights_path_ger = os.path.join(ROOT_PATH, weights_ger)

labels_path_bel = os.path.join(ROOT_PATH, "dataset_bel/labels.csv")
labels_path_ger = os.path.join(ROOT_PATH, "dataset_ger/labels.csv")
```

```
# Recuperamos los nombres de las categorias. Los diferentes tipo de señales
# que se van a clasificar.
titles_bel = traffic.read_csv(labels_path_bel, ",")
titles_ger = traffic.read_csv(labels_path_ger, ",")
```

```
def get_keras_model_bel(size_categories, shape):
    # IMPLEMENTACIÓN RED NEURONAL
    # En Keras la envoltura para cualquier red neuronal se crea con la clase Sequential
    model = models.Sequential()

    model.add(layers.Conv2D(32, (5, 5),
                            activation = 'selu', input_shape = shape))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(64, (5, 5), activation='selu'))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(128, (5, 5), activation='selu'))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Flatten())
    model.add(layers.Dense(size_categories, activation='softmax'))
    return model

def load_trained_model_bel(weights_path, size_categories, shape):
    model = get_keras_model_bel(size_categories, shape)
    model.load_weights(weights_path)
    return model
```

```
def get_keras_model_ger(size_categories, shape):
    # IMPLEMENTACIÓN RED NEURONAL
    # En Keras la envoltura para cualquier red neuronal se crea con la clase Sequential
    model = models.Sequential()

    model.add(layers.Conv2D(32, (5, 5),
                            activation='selu', input_shape=shape))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(64, (5, 5), activation='selu'))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Flatten())
    model.add(layers.Dense(size_categories, activation='softmax'))
    return model

def load_trained_model_ger(weights_path, size_categories, shape):
    model = get_keras_model_ger(size_categories, shape)
    model.load_weights(weights_path)
    return model
```

```
model_bel = load_trained_model_bel(weights_path_bel, len(titles_bel), (64, 64, 3))
model_ger = load_trained_model_ger(weights_path_ger, len(titles_ger), (32, 32, 3))

model_bel.compile(loss="categorical_crossentropy", optimizer="rmsprop", metrics=['accuracy'])
model_ger.compile(loss="categorical_crossentropy", optimizer="rmsprop", metrics=['accuracy'])
```

```
images_test_path = [os.path.join(ROOT_PATH, "images_test/Prohibido_001.jpg"),
                    os.path.join(ROOT_PATH, "images_test/Rotonda_001.jpg"),
                    os.path.join(ROOT_PATH, "images_test/Stop_001.jpg")]
```

```
def get_array_images_test(size):
    images_test = []
    for image_path in images_test_path:
        images_test.append(traffic.load_image(image_path, size, False))
    return np.array(images_test)
```

```
images_test_bel = get_array_images_test((64, 64))
images_test_ger = get_array_images_test((32, 32))



for img in images_test_bel:
    print("Dimensiones de la imagen: {}".format(img.shape))
    print("Comprobamos que la imagen este normalizada: MIN: {} MAX: {}".format(img.min(), img.max()))
    plt.imshow(img)
    plt.show()
```

```
print("Belgium:")
class_result_bel = model_bel.predict_classes(images_test_bel)
for result_bel in class_result_bel:
    print("Clase resultante: {} y título: {}". format(result_bel, titles_bel[result_bel]))


print("Germany:")
class_result_ger = model_ger.predict_classes(images_test_ger)
for result_ger in class_result_ger:
    print("Clase resultante: {} y título: {}". format(result_ger, titles_ger[result_ger]))
```

```
Belgium:
Clase resultante: 19 y título: ['19', 'B1']
Clase resultante: 7 y título: ['7', 'A23']
Clase resultante: 24 y título: ['24', 'C21']
Germany:
Clase resultante: 5 y título: ['5', 'speed limit 80 (prohibitory)']
Clase resultante: 12 y título: ['12', 'priority road (other)']
Clase resultante: 10 y título: ['10', 'no overtaking (trucks) (prohibitory)']
```

12.12. Jupyter notebook: Test detector & clasificación

Ejecuta una pruebat con la combinación de las dos redes neuronales entrenadas: Detector y clasificador.

```
import import_ipynb
import tfm_generic_functions as traffic

# https://docs.python.org/3/library/os.html
import os

# http://scikit-image.org/docs/stable/api/api.html
import skimage

# https://docs.scipy.org/doc/numpy/reference/
import numpy as np

import keras

from keras import models

# Core Layers: https://keras.io/layers/core/
# Convolution Layers: https://keras.io/layers/convolutional/
from keras import layers

# https://matplotlib.org/api/index.html
import matplotlib

# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot
import matplotlib.pyplot as plt

import statistics
from statistics import mode
```

```
importing Jupyter notebook from tfm_generic_functions.ipynb
```

```
Using TensorFlow backend.
```

```
IMG_SHAPE = (64, 64)
ROOT_PATH = os.getcwd()

h5_det = "model_data_aug_64_64_gpu_detector.h5"
h5_cla = "model_data_aug_64_64_bel_gpu.h5"

activation_det = "elu"
activation_cla = "selu"

opt_det = "adam"
opt_cla = "rmsprop"

labels_path = os.path.join(ROOT_PATH, "dataset_bel/labels.csv")
titles = traffic.read_csv(labels_path, ",")

# Images to test
images_test_path = [os.path.join(ROOT_PATH, "images_test/Prohibido_001.jpg"),
                    os.path.join(ROOT_PATH, "images_test/Rotonda_001.jpg"),
                    os.path.join(ROOT_PATH, "images_test/Stop_001.jpg")]
```

```
def get_keras_model(classes, activation):
    # IMPLEMENTACIÓN RED NEURONAL
    # En Keras la envoltura para cualquier red neuronal se crea con la clase Sequential
    model = models.Sequential()

    model.add(layers.Conv2D(32, (5, 5),
                            activation=activation, input_shape=(IMG_SHAPE[0], IMG_SHAPE[1], 3)))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(64, (5, 5), activation=activation))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Conv2D(128, (5, 5), activation=activation))
    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.MaxPooling2D(2, 2))
    model.add(layers.Dropout(0.2))

    model.add(layers.Flatten())
    model.add(layers.Dense(classes, activation='softmax'))
    return model
```

```
def load_trained_model_detector(weights_path):
    model = get_keras_model(2, activation_det)
    model.load_weights(weights_path)
    return model

def load_trained_model_classificator(weights_path, classes):
    model = get_keras_model(classes, activation_cla)
    model.load_weights(weights_path)
    return model
```

```
model_detector = load_trained_model_detector(os.path.join(ROOT_PATH, h5_det))
model_detector.compile(loss="categorical_crossentropy", optimizer=opt_det, metrics=['accuracy'])

model_classificator = load_trained_model_classificator(os.path.join(ROOT_PATH, h5_cla), len(titles))
model_classificator.compile(loss="categorical_crossentropy", optimizer=opt_cla, metrics=['accuracy'])
```

```
def get_tiles(image, tile_shape, offset):
    tiles = []
    xx = tile_shape[0]
    yy = tile_shape[1]
    for x in range(0, image.shape[0]- xx, offset):
        for y in range(0, image.shape[1] - yy, offset):
            tile = image[x:x + xx, y:y + yy]
            tiles.append(tile)
    return tiles
```

```
def get_normalized_image(image_path):
    image_test = skimage.data.imread(image_path)
    height = image_test.shape[0]
    width = image_test.shape[1]
    return skimage.transform.resize(image_test, (height, width), mode='constant')
```

```
signs = []
for image_path in images_test_path:
    image_test = get_normalized_image(image_path)
    image_tiles = get_tiles(image_test, IMG_SHAPE, 1)
    signs_aux = []
    for tile in image_tiles:
        np_tile = np.expand_dims(tile, axis=0)
        result = model_detector.predict_classes(np_tile)
        if result == 1:
            result_sign = model_classificator.predict_classes(np_tile)
            signs_aux.append(result_sign[0])
    print("Tiles: {} - valid tiles: {} Moda: {} Sign {}".format(len(image_tiles), len(signs_aux), mode(signs_aux), titl
    signs.append(signs_aux)
```

```
Tiles: 60976 - valid tiles: 7588 Moda: 22 Sign ['22', 'C1']
Tiles: 60976 - valid tiles: 5750 Moda: 35 Sign ['35', 'D1b']
Tiles: 61182 - valid tiles: 4048 Moda: 22 Sign ['22', 'C1']
```

12.13. Img2tiles.py

Se trata de un script escrito en Python. El objetivo es generar una serie de imágenes aleatorias de una determinada dimensión dado un conjunto de imágenes. Se ha programado para poder obtener un conjunto de imágenes que no tuviesen señales de tráfico. Se adjunta el código fuente en la Ilustración 27.

```python
from random import randint
import os
import sys
import skimage
from skimage.io import imsave

# Genera un número aleatorio entre 1 y max.
def getRandomPoint(max):
    return randint(1, int(max))

##### Constantes para el funcionamiento del programa
# Carpeta donde se encuentran las imágenes completas
ROOT_FOLDER = os.path.abspath("/Users/togohi/Desktop/dataset_3")
# Dimensiones de los extractos a generar
SIZE = (64, 64)
# Número de extractos aleatorios que se generarán por foto
RANDOM_BY_PHOTO = 50

# Se comprueba que existe la carpeta de entrada
if not os.path.exists(ROOT_FOLDER):
    print("ROOT Folder {} doesn't exists".format(ROOT_FOLDER))
    sys.exit(0)

# Se comprueba que existe la carpeta de salida (sino se crea)
OUTPUT_FOLDER = os.path.join(ROOT_FOLDER, "Output")
if not os.path.exists(OUTPUT_FOLDER):
    os.mkdir(OUTPUT_FOLDER)

# Se inicia una iteración por cada fichero de la carpeta de entrada
for f in os.listdir(ROOT_FOLDER):
    # Solo se tiene en cuenta las imágenes con extensión .ppm
    if f.endswith(".ppm"):
        file_path = os.path.join(ROOT_FOLDER, f)
        filename, file_extension = os.path.splitext(f)
        print(filename)

        # Cargamos la fotografía
        image = skimage.data.imread(file_path)
        print("Dimensiones: {}".format(image.shape))

        # Se extraen las dimensiones originales de la fotográfia.
        height = image.shape[0]
        width = image.shape[1]

        # Se generar de 1 a N extracciones aleatorias
        for i in range(1, RANDOM_BY_PHOTO):
            x_rand = getRandomPoint(width - SIZE[0])
            y_rand = getRandomPoint(height - SIZE[1])
            print("Punto Aleatorio: {}, {}".format(x_rand, y_rand))
            split = image[y_rand:y_rand + SIZE[1], x_rand:x_rand + SIZE[0]]
            # Se guarda en disco el extracto.
            imsave(os.path.join(OUTPUT_FOLDER, "{}_split_{}.ppm".format(filename, i)), split)
```

Ilustración 27. código fuente del extractor aleatorio de imágenes.

12.14. Bash script: Test con imageAI

Script que instala las librerías necesarias para poder ejecutar las pruebas con la librería imageAI. En la Tabla 21 se adjunta el script completo

```
#!/bin/bash

pip3 install --upgrade tensorflow
pip3 install numpy
pip3 install scipy
pip3 install opencv-python
pip3 install pillow
pip3 install matplotlib
pip3 install h5py
pip3 install keras
pip3 install https://github.com/OlafenwaMoses/ImageAI/releases/download/2.0.2/imageai-2.0.2-py3-none-any.whl
```

Tabla 21. Bash script para instalar imageAI y sus dependencias

12.15. Test con imageAI: Video

En la Ilustración 28 se adjunta el código básico para la detección de objetos sobre un video.

```
1  from imageai.Detection import VideoObjectDetection
2  import os
3
4
5  execution_path = os.getcwd()
6
7  detector = VideoObjectDetection()
8  detector.setModelTypeAsYOLOv3()
9  detector.setModelPath(os.path.join(execution_path , "yolo.h5"))
10 detector.loadModel()
11
12 input_file = os.path.join( execution_path, "video_test.mp4")
13 output_file = os.path.join(execution_path, "video_test_detected")
14 fps = 60
15 video_path = detector.detectObjectsFromVideo(input_file_path = input_file,
16                                               output_file_path = output_file,
17                                               frames_per_second = fps,
18                                               log_progress=True)
19 print(video_path)
```

Ilustración 28. Código necesario para la detección de objetos sobre un video

13. Bibliografía

[1] Docker, «Docker,» [En línea]. Available: https://www.docker.com. [Último acceso: Diciembre 2018].

[2] «VirtualEnv,» [En línea]. Available: https://virtualenv.pypa.io. [Último acceso: Diciembre 2018].

[3] J. Torres, Deep Learning Introducción práctica con Keras, Barcelona: Amazon, 2018.

[4] Wikipedia, «Aprendizaje automático,» [En línea]. Available: https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico. [Último acceso: Marzo 2019].

[5] P. N. Otero, Redes neuronales aplicadas al criptoanálisis del Advanced Encryption Standard., UOC, 2018.

[6] Pyhton, «Python,» [En línea]. Available: https://www.python.org. [Último acceso: Abril 2019].

[7] NumPy, «NumPy,» [En línea]. Available: http://www.numpy.org. [Último acceso: Abril 2019].

[8] SciPy, «SciPy,» [En línea]. Available: https://www.scipy.org. [Último acceso: Abril 2019].

[9] Matplotlib, «Matplotlib,» [En línea]. Available: https://matplotlib.org. [Último acceso: Abril 2019].

[10] PYPL, «PYPL PopularitY of Programming Language,» [En línea]. Available: http://pypl.github.io. [Último acceso: Abril 2019].

[11] Keras, «Keras,» [En línea]. Available: https://keras.io. [Último acceso: Abril 2019].

[12] P. Jupyter, «Project Jupyter,» [En línea]. Available: https://jupyter.org. [Último acceso: Abril 2019].

[13] Tensorflow, «Tensorflow,» [En línea]. Available: https://github.com/tensorflow/tensorflow. [Último acceso: Abril 2019].

[14] Theano, «Theano,» [En línea]. Available: https://github.com/Theano/Theano. [Último acceso: Abril 2019].

[15] CNTK, «CNTK,» [En línea]. Available: https://github.com/Microsoft/cntk. [Último acceso: Abril 2019].

[16] CUDA, «CUDA,» [En línea]. Available: https://www.nvidia.es/object/cuda-parallel-computing-es.html. [Último acceso: Abril 2019].

[17] T. G. T. S. D. Benchmark, «The German Traffic Sign Detection Benchmark,» [En línea]. Available: http://benchmark.ini.rub.de/?section=gtsdb&subsection=dataset.

[18] B. Dataset, «BelgiumTS Dataset,» [En línea]. Available: https://btsd.ethz.ch/shareddata/.

[19] Agonzalezhidalgo, «Repositorio GitHub - Dataset German,» [En línea]. Available: https://github.com/agonzalezhidalgo/dataset_ger.

[20] Agonzalezhidalgo, «Repositorio Github - Belgium,» [En línea]. Available: https://github.com/agonzalezhidalgo/dataset_bel.

[21] T. G. T. S. D. Benchmark, «German Traffic Sign Detection Benchmark GTSDB,» [En línea]. Available: https://sid.erda.dk/public/archives/ff17dc924eba88d5d01a807357d6614c/published-archive.html.

[22] Agonzalezhidalgo, «GitHub Dataset Detector,» [En línea]. Available: https://github.com/agonzalezhidalgo/dataset_detector.

[23] «Activations - Keras Documentation,» [En línea]. Available: https://keras.io/activations/. [Último acceso: Abril 2019].

[24] «Optimizers - Keras documentation,» [En línea]. Available: https://keras.io/optimizers/. [Último acceso: Abril 2019].

[25] Keras, «Core Layers,» [En línea]. Available: https://keras.io/layers/core/#flatten. [Último acceso: Mayo 2019].

[26] Keras, «Image Preprocessing - Keras,» [En línea]. Available: https://keras.io/preprocessing/image/. [Último acceso: Mayo 2019].

[27] Apple, «Core ML - Apple Developer,» [En línea]. Available: https://developer.apple.com/documentation/coreml#overview. [Último acceso: Mayo 2019].

[28] Apple, «Converting Trained Models to Core ML,» [En línea]. Available: https://developer.apple.com/documentation/coreml/converting_trained_models_to_core_ml. [Último acceso: Mayo 2019].

[29] Nvidia, «NVIDIA GeForce GT 650M,» [En línea]. Available: https://www.geforce.com/hardware/notebook-gpus/geforce-gt-650m. [Último acceso: Mayo 2019].

[30] «Ubuntu,» [En línea]. Available: https://www.ubuntu.com. [Último acceso: Mayo 2019].

[31] NVidia, «CUDA GPUs,» [En línea]. Available: https://developer.nvidia.com/cuda-gpus. [Último acceso: Mayo 2019].

[32] Google, «Google Colaboratory,» [En línea]. Available: https://colab.research.google.com/. [Último acceso: Mayo 2019].

[33] «YOLO: Real-time object detection,» [En línea]. Available: https://pjreddie.com/darknet/yolo/. [Último acceso: Mayo 2019].

[34] «Keras RetinaNet,» [En línea]. Available: https://github.com/fizyr/keras-retinanet. [Último acceso: Mayo 2019].

[35] «ImageIA,» [En línea]. Available: https://github.com/OlafenwaMoses/ImageAI. [Último acceso: Mayo 2019].

[36] A. González, «Recorrido test,» [En línea]. Available: https://www.youtube.com/watch?v=J-mKbDS03iE. [Último acceso: Mayo 2019].

[37] A. González, «Recorrido test con ImageIA,» [En línea]. Available: https://www.youtube.com/watch?v=f9Ps3yTZoaA. [Último acceso: Mayo 2019].

[38] Docker, «Docker | Hub,» [En línea]. Available: https://hub.docker.com.

[39] agonzalezhidalgo, «Traffic | Docker Hub,» [En línea]. Available: https://cloud.docker.com/repository/docker/agonzalezhidalgo/traffic.

[40] Agonzalezhidalgo, «GitHub - Repositorio,» [En línea]. Available: https://github.com/agonzalezhidalgo. [Último acceso: Mayo 2019].

[41] Wikipedia, «Stochastic gradient descent,» [En línea]. Available: https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[42] Keras, «Core Layers,» [En línea]. Available: https://keras.io/layers/core/#dropout. [Último acceso: Mayo 2019].