Universidad Complutense de Madrid
Facultad de Informática

Reconocimiento de objetos en imágenes mediante técnicas de aprendizaje profundo: Redes Neuronales Convolucionales

Jaime Molinero Lacave
Andrei Ionut Vaduva

Trabajo de fin de grado en Ingeniería Del Software
Curso 2018/2019
Director: Gonzalo Pajares Martinsanz

ÍNDICE

LISTADO DE FIGURAS

LISTADO DE TABLAS

RESUMEN

El presente trabajo describe el diseño y desarrollo de una aplicación para el reconocimiento en tiempo real de objetos en imágenes digitales mediante técnicas de aprendizaje profundo (*Deep Learning*), más concretamente con Redes Neuronales Convolucionales (RNC). Los objetos a reconocer pueden ser de cualquier tipo, si bien en este trabajo se ha orientado hacia el reconocimiento de una plataforma para el aterrizaje seguro de un vehículo aéreo no tripulado VANT (UAV, Unmanned Aerial Vehicle).

La imagen, obtenida por un sistema de captura, se transmite a un servidor capaz de procesar las imágenes utilizando los modelos RNC ya creados y pre-entrenados para responder acorde a la información recibida. Estos modelos se re-entrenan con imágenes propias. Por tanto, el procesamiento de las imágenes consta de las dos fases siguientes:

- Uso de un modelo pre-entrenado, "*MobileNet"*, especialmente diseñado para su ejecución de manera eficiente en dispositivos integrados, siendo capaz de distinguir entre mil clases de objetos diferentes para realizar una primera predicción.

- A partir del modelo pre-entrenado y con los pesos de las capas de convolución disponibles, se suministran imágenes específicas, objeto del reconocimiento, con el fin de llevar a cabo un proceso de re-entrenamiento, que finaliza con la actualización de los pesos en las mencionadas capas de convolución.

La aplicación consiste en un diseño flexible con un interfaz amigable e intuitiva, que ofrece la posibilidad de ser utilizada sin conocimientos específicos en las tecnologías empleadas. Gracias a esta flexibilidad, es posible utilizar la plataforma para su uso en el reconocimiento de distintos tipos de objetos sin más que realizar el re-entrenamiento con un número relativamente bajo de imágenes que los contienen.

PALABRAS CLAVE

- Visión Artificial.
- Redes Neuronales Convolucionales.
- Plataforma de aterrizaje.
- VANT.
- Tensorflow.
- Librería.
- JavaScript.
- API REST.
- Modelos.
- Clasificación.

ABSTRACT

The present work describes the design and development of an application for the recognition in real time of objects in digital images by means of deep learning techniques (Deep Learning), more specifically with Convolutional Neural Networks (CNN). The objects to be recognized can be of any type, but this work has been orientated towards the recognition of a platform for the safe landing of an unmanned aerial vehicle (UAV).

The image, obtained by a capture system, is transmitted to a server capable of processing the images using the CNN models already created and pre-trained to respond according to the information received. These models are re-trained with their own images. Therefore, the processing of the images consists of the following two phases;

- Use of a pre-trained model, “MobileNet”, especially designed for an efficient execution in integrated devices, being able to distinguish between a thousand different object classes to make a first prediction.

- From a pre-trained model and with the weights of the available convolution layers, specific images are provided, object of the recognition, in order to carry out a re-training process, which ends with the updating of the weights in the convolution layers.

The application consists of a flexible design with a friendly and intuitive interface, which offers the possibility of being used without specific knowledge in the technologies used. Thanks to this flexibility, it is possible to use the platform for use in recognition of different types of objects without further re-training with a number relatively low of images that contain them.

KEYWORDS

- Artificial Vision.
- Convolutional Neural Network.
- Landing platform.
- UAV.
- Tensorflow.
- Framework.
- JavaScript.
- API REST.
- Models.
- Classification.

LISTA DE ABREVIATURAS

- **IA** Inteligencia Artificial.
- **IC** Ingeniería del Conocimiento.
- **VANT** Vehículo Aéreo No Tripulado.
- **UAV** Unmanned Aerial Vehicle.
- **GPS** Sistema de Posicionamiento Global (en inglés, Global Positioning System).
- **API** Interfaz de Programación de Aplicaciones (en inglés, Application Programming Interface).
- **REST** Transferencia de Estrado Representacional (en inglés, Representational State Transfer).
- **UWP** Universal Windows Platform.
- **CPU** Central Processing Unit (Unidad Central de Procesamiento).
- **GPU** Graphics Processing Unit (Unidad de Procesamiento Gráfico).
- **SPA** Single Page Application.
- **IDE** Integrated Development Environment.
- **TPUs** Cloud Tensor Processing Units.
- **URL** Uniform Resource Location.

1 INTRODUCCIÓN

1.1 PRELIMINARES

En los últimos años, con el avance metodológico y tecnológico, se han desarrollado novedosas y distintas aplicaciones cuya utilidad en distintos ámbitos es más que notable. Dentro de ellas destacan los métodos desarrollados bajo el paradigma del aprendizaje profundo (*Deep Learning*) y más concretamente las denominadas Redes Neuronales Convolucionales (RNC).

Resultan conocidas en este sentido las aplicaciones desarrolladas en el mundo del transporte inteligente y como ayuda a la conducción en vehículos automatizados, actualmente en evolución hacia su total autonomía, donde estas mismas técnicas están llamadas a jugar un papel ciertamente relevante. En este ámbito destacan aplicaciones orientadas a la detección de obstáculos, peatones, señales de tráfico en las vías y las líneas delimitadoras de los carriles de circulación, como se puede ver por ejemplo en los últimos modelos de los vehículos de grandes marcas como pueden ser Mercedes Benz con su sistema PRE-SAFE (2019), el cual detecta los posibles peligros de tal manera que intenta paliarlos o en caso de no ser posible, de preparar el entorno para que el daño causado a los ocupantes sea el menor posible.

Es evidente que su aplicación aumenta la seguridad vial en todos los ámbitos, con posibilidades como que los vehículos inteligentes dotados de estas tecnologías asuman, en determinados momentos o de forma continuada, decisiones propias. Por ejemplo esto se puede observar en los últimos modelos de Tesla con la incorporación de Tesla Autopilot (2019), que permite liberar al conductor de la tarea de la conducción, cuando éste lo requiera, otorgándole al vehículo una autonomía casi completa en la conducción, siendo capaz de seguir por sí mismo la ruta establecida y detectando todos los obstáculos necesarios para no provocar daños ni incumplir la legalidad. Esta autonomía es casi completa debido a que obligan al conductor a mantenerse atento detectando por ejemplo si sus manos están sujetando el volante, por si en algún momento es necesaria su intervención.

Estas nuevas aplicaciones, también han propiciado avances en muchos ámbitos de la seguridad, no sólo en la referente a la seguridad vial; sino por ejemplo, en la seguridad en los accesos a las propiedades como pueden ser vehículos, fincas, fábricas, recintos vigilados, etc. un simple dispositivo móvil, una huella dactilar o una videocámara capaz de detectar el rostro de los usuarios, con el único requisito de tener acceso a servidores remotos mediante los sistemas de comunicación actuales, y en un futuro próximo haciendo uso de las redes 5G las cuales otorgarán aún más rapidez en las respuestas.

Otro ámbito de aplicación de estas tecnologías viene del mundo de los Vehículos Aéreos No Tripulados (VANT) (UAV, Unmanned Aerial Vehicles) conocidos también como drones. Los VANT están equipados con diversos dispositivos tecnológicos, destacando, cámaras digitales a color e incluso multiespectrales, GPS, sensores de presión, altura, temperatura entre otros posibles (Pajares, 2015). La combinación de todos estos dispositivos permite obtener información de su entorno, procesarla y tomar decisiones en consecuencia. Las RNC tienen mucho que decir en este ámbito, de forma que la dotación de una mayor inteligencia les proporciona un mayor nivel de autonomía.

Es bien conocido el uso de los VANT en el comercio electrónico, donde la entrega de pedios en el menor tiempo posible y de forma segura, se ha convertido en un reto de gran relevancia a través de dichos vehículos. Las empresas logísticas y de distribución están apostando fuerte en esta línea. No obstante, con el incremento de la demanda a estas empresas, aparecen problemas que antes no existían como, por ejemplo, la entrega de pedidos en zonas geográficas complejas, en tiempos muy cortos o en franjas horarias no compatibles con los receptores. A la hora de realizar estos envíos, como se ha indicado previamente, el tiempo es fundamental e impacta directamente en el negocio y reputación. Por ello, estas empresas deciden invertir en nuevas tecnologías que permitan realizar los envíos de forma rápida y con el menor coste posible.

Los VANT, dentro de sus limitaciones de autonomía y carga, tienen capacidad de acceso a zonas remotas, realizar las rutas por el aire y sin tripulación, lo que implica ahorros en personal, en tiempos de reparto, en vehículos y combustibles propiciando además una mejora para el medio ambiente. Todo ello orientado a conseguir el mayor nivel de eficiencia en la entrega de los pedidos. En este sentido, es bien conocido que los VANT tienen planificada una ruta determinada, guiándose mediante señal GPS, hasta el punto de entrega. Si bien, existe un problema importante justo en el momento de la entrega, que consiste en determinar el punto exacto donde depositar el objeto, teniendo en cuenta que éste debe ser seguro. Debido a que en determinadas circunstancias la precisión del GPS puede ser un obstáculo, una posible solución pasa por reconocer una zona de terreno o área urbana que asegure la entrega sin riesgo mediante la utilización de técnicas de reconocimiento de texturas en imágenes tomadas con la cámara a bordo. No obstante, esto entraña un riesgo importante, en el sentido de que dichas zonas pueden no ser identificadas de forma correcta. En este sentido, otra solución consiste en el reconocimiento de una plataforma, diseñada específicamente para tal fin, de suerte que se trate de un objeto exclusivo no presente en los entornos de operación. Es aquí precisamente donde se enfoca el presente trabajo, de forma que mediante las técnicas basadas en las RNC se llegue a la identificación de la misma. Para incrementar la seguridad en las entregas cabe la posibilidad de que para cada pedido se genere un diseño de plataforma único, que evite ambigüedades en el reconocimiento.

Cabe señalar, además, que este planteamiento no es exclusivo para el caso de las entregas logísticas indicadas, sino que es bien conocido cómo en ocasiones los VANT deben posarse para operaciones de recarga de baterías una vez agotado el periodo de autonomía. Los vehículos autónomos, en el futuro deberán estacionar en plazas de aparcamiento específicas, a veces en entornos donde la señal GPS no llega. En consecuencia, en estos casos la disposición de una plataforma identificativa para tales propósitos resulta de vital importancia. Las técnicas desarrolladas en el presente proyecto son válidas también en estos casos concretos.

Por tanto, volviendo al objetivo del presente trabajo y dado que ya existen trabajos sobre reconocimiento de plataformas de aterrizaje para VANT (Piñeiro 2016; Arroyo y col., 2017; Garcia-Pulido y col., 2017) se ha determinado que una solución factible al problema planteado consiste en la generación de una sencilla disposición de figuras geométricas coloreadas, que pueden imprimirse sobre un folio de tamaño A4, acción ésta que puede realizar el propio destinatario de los servicios de entrega solicitados. De esta forma, el usuario coloca la plataforma en el suelo, proporcionando las coordenadas geográficas, de forma que el VANT, al detectar la plataforma, aterrizaría o depositaría el pedido sobre la misma. La diferencia principal con respecto a los diseños presentados en las anteriores referencias estriba en el uso de las RNC.

1.2 MOTIVACIÓN

Como se ha indicado previamente, la motivación principal que origina el presente trabajo estriba en la necesidad detectada de desarrollar métodos inteligentes para el reconocimiento de objetos, en este caso una plataforma de aterrizaje mediante RNC.

Por otra parte, dado el auge y el avance tecnológico, es conveniente y necesario desplegar una herramienta flexible que proporcione una capacidad tecnológica para futuros desarrolladores en este tipo de aplicaciones.

Además, situándose en el punto de vista de potenciales usuarios, resulta de gran importancia el hecho de disponer de una herramienta con capacidad de proporcionar una visión sobre el desempeño de las metodologías aplicadas para el reconocimiento de cualquier tipo de objetos en imágenes, no sólo plataformas, de forma que se ofrezcan y abran posibilidades de negocio en cualquier ámbito, no sólo mediante el uso de VANT.

Finalmente, hay que indicar que desde el punto de vista del proceso enseñanza-aprendizaje, el hecho de desarrollar una aplicación de estas características supone un reto de relevancia en lo que respecta a la adquisición de conocimiento en el ámbito de las tecnologías de vanguardia en claro auge y progreso.

1.3 OBJETIVOS

El objetivo general de este proyecto es el desarrollo de una aplicación para la identificación de una plataforma de aterrizaje para un VANT mediante imágenes capturadas con la videocámara acoplada al vehículo.

Además de este objetivo principal, se identifican los siguientes objetivos específicos:

1. Realizar una revisión de técnicas, algoritmos y software ya disponibles en el ámbito de la Inteligencia Artificial y el reconocimiento de imágenes.

2. Aplicar técnicas de Aprendizaje Profundo, mediante RNC, para el reconocimiento de la plataforma.

3. Crear una aplicación flexible y amigable que permita realizar el proceso de gestión de modelos desde su creación y entrenamiento hasta su uso en la detección de plataformas de aterrizaje para los VANT.

4. Realizar pruebas experimentales para la validación de los desarrollos, con el fin de determinar la viabilidad del modelo con la mayor precisión posible.

En cuanto al plan de trabajo, todas las fases del proyecto se han llevado a cabo semanalmente, pero no en su totalidad, por lo que el tiempo dedicado a cada objetivo es aproximado, por lo tanto, se han dedicado seis semanas al primer punto de este apartado y ocho semanas a los apartados dos, tres y cuatro.

1.4 ESTRUCTURA DE LA MEMORIA

Este documento está formado por seis secciones, dos anexos, un listado de figuras, tablas y abreviaturas respectivamente y un breve resumen del proyecto.

A continuación, se indican los aspectos más importantes de cada sección y anexo.

1. **Introducción:** En esta sección se puede encontrar un resumen detallado y general de los aspectos más importantes que se van a tratar en este documento, es decir, la finalidad principal de este proyecto, la motivación y los objetivos además de las contribuciones personales de cada miembro del equipo.

2. **Revisión de metodologías:** Este apartado trata con detalle metodologías, técnicas y conceptos relacionados con la Inteligencia Artificial, en concreto se realiza una introducción mediante trabajos relacionados con este proyecto para dar paso a una descripción del concepto de redes neuronales convolucionales y a al concepto de clasificación con imágenes. En el último apartado de esta sección se encuentran conceptos relacionados con la librería que ha permitido llevar a cabo el proyecto.

3. **Análisis y diseño:** En este punto se describe la labor de investigación y documentación sobre diferentes tecnologías, librerías, plataformas, APIs, así como las pruebas de concepto realizadas en cada caso describiendo las ventajas y desventajas de dichas tecnologías. En el último apartado de esta sección se realiza una descripción detallada de la implementación de este proyecto, así como de las tecnologías utilizadas y la gestión del proyecto por parte del equipo.

4. **Análisis de resultados:** Una vez construida la aplicación, se han creado varios modelos para su posterior entrenamiento y prueba. Este apartado muestra los resultados obtenidos de dichos entrenamientos y pruebas en función de los datos de entrada.

5. **Conclusiones y Trabajo Futuro:** Antes y durante el desarrollo de este proyecto surgen ideas para la mejora de la aplicación que no se han podido poner en práctica por diversos motivos, entre ellos la definición de alcance del proyecto. También encontramos la conclusión obtenida al finalizar el proyecto.

6. **Bibliografía:** Para llevar a cabo este proyecto se ha utilizado información procedente de orígenes varios. En este apartado podemos encontrar todas las referencias a dichos orígenes de información.

7. **Anexo I:** Consiste en un manual detallado para realizar la correcta instalación y ejecución de la aplicación.

8. **Anexo II:** Es un manual de usuario para saber cómo hay que utilizar la aplicación, desde la creación de una categoría de modelos hasta la realización de una clasificación.

1.5 CONTRIBUCIONES PERSONALES

Jaime Molinero Lacave

Antes de comenzar este proyecto mi idea era realizar un proyecto de fin de grado que fuera entretenido y que me permitiera experimentar y mejorar los conocimientos adquiridos durante mi paso por la Facultad de Informática de la Universidad Complutense de Madrid.

Me asustaba la idea de realizar durante un curso entero un proyecto que no me entusiasmara y que fuera a realizar sin ningún tipo de motivación. Por suerte, un día en la clase de Ingeniería del Conocimiento (IC) la cual era impartida por el profesor D. Gonzalo Pajares, éste comentó que disponía de un proyecto de fin de grado sobre inteligencia artificial y aterrizaje autónomo de drones que me llamó la atención.

Cuando comentó brevemente la idea, me fascinó y terminado el curso, procedí a hablar con él para ver cuál era el enfoque que le pretendía dar y si había libertad para experimentar con tecnologías nuevas. Dado que el profesor no puso ninguna pega a esta condición, decidí comentárselo a Andrei Ionut Vaduva; ya que él se encontraba también en la misma situación y accedimos los dos a tener otra reunión con Gonzalo para que nos volviera a explicar de nuevo la idea del proyecto y aceptamos.

Una vez comentado cómo se llegó a este punto, procedo a describir cuál ha sido mi labor en este proyecto.

Durante las primeras reuniones, la idea era comentar y exponer posibles soluciones para solventar el problema que teníamos entre manos, para ello mi aportación consistió sobre todo en comentar posibles soluciones que se me venían a la cabeza a modo de *brainstorming* de tal manera que unas gustaban más y otras menos en conjunto, durante aquellas reuniones salieron muy buenas ideas, no quisimos ser del todo conscientes de la dificultad que planteaban muchas de ellas; ya que el objetivo era proponer soluciones.

Durante el proyecto estaba en esta primera etapa de análisis y reuniones, mi labor consistió también en la búsqueda de tecnologías, ejemplos y toda clase de material a gran escala que nos permitiese avanzar en este proyecto lo más rápido y mejor posible, incluyendo también experiencias de usuarios para saber con qué clase de problemas tendríamos que lidiar durante nuestro desarrollo, para de esta manera anticiparnos y ya poder contar con ellos en caso de elegir un determinado camino.

Durante la búsqueda de tecnologías descubrí temas tan interesantes como la utilización de un modelo pre-entrenado denominado "YOLO (2019)", el cual servía para poder realizar clasificaciones sobre imágenes en tiempo real con reconocimiento de objetos por regiones de forma rápida y precisa. Además, permite reconocer múltiples objetos en una misma imagen. Por desgracia, no pudimos implementar YOLO (2019) en nuestra aplicación ya que el modelo sobre el cual realizaba los entrenamientos era el modelo Darknet-53 con unas exigencias tecnológicas y conceptuales en reconocimiento de imágenes, que en principio quedaban fuera del ámbito del proyecto, ya que se hubiese requerido algún curso de especialización en la materia. A pesar de que el profesor ofrecía la posibilidad de avanzar en forma de seminarios sobre el tema.

Más adelante, comenzando con los primeros desarrollos sobre las librerías que fuimos descubriendo y probando, mi labor fue, al igual que la de Andrei, la de intentar desarrollar en conjunto una aplicación de muestra que nos permitiera realizar un entrenamiento de imágenes y etiquetas utilizando en ese momento la tecnología que fuera posible para cumplir nuestro objetivo. Para ello, nuestro primer prototipo fue realizado con una librería escrita en .Net denominada "Accord.Net (2018)".

Durante el primer desarrollo para este proyecto, cuando empezamos utilizando el *framework* Accord.Net (2018), mi trabajo consistía en encargarme tanto de ir entendiendo el funcionamiento de las redes neuronales ejecutando demostraciones sencillas, leyendo guías y viendo videos, como de crear un sistema capaz de recibir los datos necesarios para realizar el entrenamiento y la clasificación sobre los modelos generados con Accord.Net (2018).

Por tanto, durante la fase de desarrollo me encargué de definir las interfaces que comunicaban la aplicación, la arquitectura del sistema, las vistas, tanto las de la aplicación de escritorio como las de la aplicación web, ya que, al principio hicimos un desarrollo local y después un desarrollo con despliegue web. Me encargué también de establecer la infraestructura en Microsoft Azure (2018) para realizar subidas de imágenes, realicé las conexiones necesarias en la aplicación con esta infraestructura y las pruebas.

Más adelante, viendo que Accord.Net (2018) no proporcionaba los resultados esperados, comenzó una etapa en la cual comencé con búsqueda de otro *framework* que ofreciera más fiabilidad en las clasificaciones. A pesar de tener ya casi toda la infraestructura montada para Accord.Net (2018) con Microsoft .Net Framework 4.5 (2018) , no veíamos la seguridad necesaria para alcanzar una solución fiable y real al problema que pretendíamos resolver, el cual era entrenar un modelo con redes neuronales para hacer que un dron sea capaz de elegir un lugar óptimo donde aterrizar. Por lo que decidimos probar con Microsoft CNTK (2018).

Microsoft CNTK (2018) resultó bastante complicado de entender e implementar, además de la falta de ejemplos y de guías no ayudaba a ello, por lo que tuvimos que elegir entre mejorar nuestro modelo desarrollado con Accord.Net (2018) o buscar una solución alternativa a pesar de tener que empezar de nuevo.

Decidimos empezar de nuevo, y la decisión llegó a buen puerto. Por suerte, en noviembre de 2018 acudí durante un fin de semana a un evento de desarrolladores denominado "Commit Conf (Nov. 2018)" gracias a la empresa en la que trabajaba en ese momento. En este evento, asistí a una serie de conferencias y entre ellas había una que trataba sobre TensorflowJS (2019) por lo que decidí asistir a pesar de haber descartado Tensorflow (2019) anteriormente por su complejidad.

La conferencia me pareció increíblemente interesante, en ella pude observar que los interlocutores fueron capaces de entrenar un modelo para que aprendiera a jugar a un videojuego llamado "*Super Hexagon*" el cual consiste en girar, mediante las flechas del teclado, una pequeña flecha que rota sobre un pequeño hexágono ubicado en el centro de la pantalla y de esta forma intentar que la flecha esquive las paredes que se van acercando al centro dejando huecos libres para pasar. Para ello, utilizaron la versión de "*Super Hexagon*" para JavaScript, llamada "Hexagon.js (2018)" y entrenaron un modelo de TensorflowJS (2019) mediante reconocimiento de imágenes con unas diez mil muestras en las cuales indicaban la imagen de un estado del juego con la correspondiente tecla que debía pulsar en

ese momento el jugador para poder continuar jugando sin perder, siendo la tecla una clase del modelo.

Fue en esta conferencia cuando descubrí TensorflowJS (2019) y pude comprobar la potencia que tenía realmente al ver como en tan pocas líneas de código habían sido capaces de poder montar un modelo para algo que parecía muy complejo.

Además, en esta conferencia explicaron de forma muy sencilla y correcta el concepto de tensores de TensorflowJS (2019), los diferentes tipos de tensores que podíamos encontrar, qué son las variables, cuáles son las operaciones, además de los distintos tipos de funciones de activación, pérdida y de optimización que comentaron más por encima. Es decir, que gracias a esta conferencia pude llevarme gran parte del trabajo de investigación a casa y pude empezar a probar.

Una vez acabó la conferencia se lo comenté todo a Andrei y empezó la investigación sobre TensorflowJS (2019), decidiendo montar nuestro primer prototipo de aplicación para entrenar modelos con imágenes de inteligencia artificial con redes neuronales.

Durante el desarrollo de este prototipo trabajé junto a Andrei en el desarrollo del modelo, en la manera de aplicar la mejor configuración posible para que nuestro entrenamiento de imágenes tuviera mayor precisión a la hora de predecir, y cuando tuvimos un tipo de modelo que nos convencía, comenzamos el desarrollo de una aplicación que se ejecutaba completamente en el navegador.

Para realizar esta primera aplicación web encargada de entrenar modelos y de clasificar imágenes, me encargué junto a Andrei de escoger los *frameworks* que íbamos a utilizar, por lo que, a la hora de elegir *frameworks* de arquitectura MVC para desarrollo de aplicaciones web, decidí utilizar la última versión de Angular 6 (2019) disponible en ese momento. Si bien, para otorgar rapidez al desarrollo y hacerlo más sencillo de mantener, finalmente se optó por utilizar React (2019) ya que, con este *framework* todos los miembros del equipo teníamos el conocimiento necesario para hacer cualquier modificación cuando fuera, debido a que ambos teníamos experiencia al haberlo utilizado en nuestro entorno profesional.

Cuando culminamos el sprint en el que implementamos todo lo que creíamos conveniente para el prototipo, decidimos participar en una reunión con el profesor para mostrarle lo que habíamos implementado. El software que habíamos desarrollado le había gustado, aunque Andrei y yo decidiéramos ir a más posteriormente.

Utilizar TensorflowJS (2019) en el navegador estaba muy bien, pero podíamos aplicar aún más los conocimientos obtenidos durante nuestro último curso y decidimos montar un servidor con Express (2019) sobre NodeJS (2019) del que se encargó Andrei. Además, Andrei se encargó de portar la parte rescatable de nuestro antiguo código frontal, que se encargaba de todo lo relacionado con TensorflowJS (2019) en el navegador al servidor Express (2019) nuevo.

Por mi parte, me dediqué a realizar un nuevo frontal con React (2019), rescatando algunos módulos antiguos reutilizables y preparando la aplicación para una comunicación cliente servidor de tal manera que toda la comunicación con TensorflowJS (2019) y el procesamiento se realizase en servidor. Además, durante este desarrollo hice hincapié en el estilo de la aplicación para que quedase más vistosa y fuera más sencilla de utilizar que la versión anterior.

Con este nuevo desarrollo, pudimos tener completamente abstraída la lógica del negocio de la lógica de la vista, y podíamos tener la posibilidad en cualquier momento de implementar diferentes frontales para montar múltiples entornos distintos. También, con esta nueva versión de la aplicación ganaríamos rapidez en los entrenamientos y clasificaciones al poder tener TensorflowJS (2019) ejecutándose en un servidor y no utilizando los recursos del navegador web, que son mucho más limitados, obteniendo ahora la posibilidad de hacer entrenamientos utilizando mayor potencia de CPU u otorgando la posibilidad incluso de utilizar la GPU.

Para culminar, a continuación resumo mi participación en este proyecto como una participación basada en el estudio de diferentes tecnologías de inteligencia artificial, estudio de diferentes tipos de modelos, no solo de aquellos basados en imágenes, sino también en modelos basados en texto y datos; estudios sobre el procesamiento de imágenes, mejoras optimización, múltiples refactorizaciones del código del proyecto para hacerlo más mantenible y legible, establecer arquitecturas del modelo en conjunto a mi compañero, arquitectura de las aplicaciones, arquitectura de sistemas y sobre todo un constante aprendizaje; por lo cual, con este proyecto siento que he cumplido los objetivos que me propuse al inicio.

Mis primeras contribuciones a este proyecto consistieron en una serie de reuniones con Gonzalo Pajares y Jaime Molinero para definir el inicio del proyecto y los pasos a seguir.

Durante estas reuniones se habló de las posibles tecnologías a utilizar y del objetivo del proyecto. En concreto se acordó la necesidad de realizar una labor de investigación de las diferentes librerías existentes relacionadas con la Inteligencia Artificial en el ámbito del aprendizaje profundo, comenzando con el modelo AlexNet (2018) (ImageNet, 2018).

Después de estas reuniones comencé a investigar las diferentes librerías y aplicaciones ya disponibles, relacionadas con la Inteligencia Artificial. El primer paso fue buscar AlexNet, (2018) pero tras la investigación, decidimos que dicha librería no tenía la compatibilidad necesaria con la aplicación que querríamos desarrollar, por lo que busqué otra librería.

Durante ese intervalo de tiempo de búsqueda, me encontraba realizando las prácticas curriculares en una empresa que tenía convenio con la Universidad Complutense de Madrid. Al tener la empresa conocimiento del objetivo de mi Trabajo de Fin de Grado, me propusieron un proyecto de I+D interno.

Dicho proyecto se llamaba “Piedra, Papel y Tijeras”. El objetivo de este proyecto era crear una aplicación que mediante la Inteligencia Artificial fuera capaz de reproducir el juego del mismo nombre. Para llevar a cabo esta aplicación me indicaron utilizar Microsoft Cognitive Services (2018) que dispone de una sección llamada “Microsoft Custom Vision (2018)”.

Durante el desarrollo de esta aplicación aprendí a utilizar la API de visión de Microsoft Cognitive Services (2018), que básicamente lo que permite es crear un proyecto de clasificación o detección de objetos en la nube. Al crear el proyecto el siguiente paso es la creación de unas etiquetas que se utilizan para identificar a cada tipo de imagen, es decir, cada etiqueta creada representaría una clase del modelo.

En la asignatura Ingeniería del Conocimiento de cuarto curso se utilizan algunos algoritmos de clasificación, pero no de detección de objetos, por lo que en primer lugar decidí que la mejor forma de llevar a cabo el proyecto “Piedra, Papel o Tijera” era mediante la creación de un proyecto de clasificación, ya que me resultaba más familiar y tenía más conocimientos sobre su funcionamiento.

Al crear el proyecto de clasificación de imágenes, creé tres etiquetas, una para cada opción del mencionado juego, entrenado dicho modelo con quinientas imágenes para cada clase, es decir, alrededor de mil quinientas muestras para el modelo.

Una vez creada la API alojada en la nube de Microsoft (incluidas las imágenes, que después de realizar el entrenamiento permanecían guardadas) desarrollé un proyecto *“Front-End”* que además de contener la lógica del juego, contenía la llamada a la API previamente creada y entrenada.

Los resultados de las clasificaciones para este proyecto no eran muy convincentes, la API al realizar clasificaciones de tipo “Tijera” y tipo “Piedra” tendía a confundir “Tijera” con “Piedra”. Analizados estos resultados, llegue a la siguiente conclusión: “Se necesitan más muestras, al ser un proyecto de clasificación y al ser la entrada de datos de tipo imagen, es decir, una entrada compleja, con muchas variables”.

El siguiente paso fue crear una API de Microsoft Custom Vision (2018), pero esta vez el proyecto sería de detección de objetos. La creación de esta API fue exactamente igual que la anterior a nivel de proyecto, la diferencia entre ambos se produce a la hora de introducir las imágenes ya que en un proyecto de clasificación solo hay que seleccionar la clase correspondiente a la imagen, pero en un proyecto de detección de objetos, además de seleccionar la clase a la que pertenece la muestra, hay que señalar la región en la que se encuentra el objeto que se quiere detectar dentro de la imagen.

Para poder realizar el entrenamiento de este modelo, añadí cuarenta imágenes para cada clase, que como se ha dicho son: “Piedra”, “Papel” y “Tijera”.

Al analizar los resultados de las predicciones de este proyecto, se generaron muchos menos fallos que en el proyecto de clasificación, los resultados de las predicciones eran mucho más precisos con muchas menos muestras.

Después de realizar todo este trabajo tuve una reunión con Gonzalo Pajares para comentarle el posible uso de estos servicios de Microsoft en el Trabajo de Fin de Grado.

En esta reunión decidimos que el primer paso para poder utilizar estos tipos de servicios era ver qué tipo de arquitectura se estaba utilizando, si se estaba utilizando técnicas de “*Machine Learning*” o de “*Deep Learning*”, en cuyo caso habría que investigar el tipo de redes neuronales que admitía.

Intenté obtener documentación sobre los procesos internos que se llevan a cabo en el servicio de Microsoft Cognitive Services (2018) sin éxito, al ser una API privada y en algunos aspectos en desarrollo, por lo que no se pudo utilizar para llevar a cabo el presente proyecto.

Al no poder utilizar estos servicios, por lo comentado anteriormente, proseguimos la labor de búsqueda de una librería que cumpliera los requisitos necesarios.

Durante esta nueva fase de búsqueda encontré Tensorflow (2019), una plataforma de código abierto para *Machine Learning* que dispone de varias herramientas y librerías para la construcción y el despliegue de aplicaciones que utilizan *Machine Learning*. Además, está respalda por una amplia comunidad de desarrolladores.

Tras leer la documentación, creía que esta librería cumplía con todos los requisitos, pero para asegurarme intenté desarrollar una pequeña aplicación, tarea que me resultó bastante complicada ya que en esas fechas aún no había aprendido Python.

Al no conseguir llevar a buen puerto dicha aplicación decidí reunirme con Gonzalo para explicarle la situación y la posibilidad de utilizar Tensorflow (2019) en el desarrollo del proyecto ya que tenía previsto cursar una asignatura para aprender Python.

Finalmente se decidió proseguir con la búsqueda de librerías ya que Tensorflow (2019) es una plataforma muy potente orientada a profesionales, con muchas posibles configuraciones y requiere de conocimientos avanzados.

La siguiente librería que encontré fue Accord.Net (2018), *framework* de *Machine Learning* de código abierto combinado con librerías de procesamiento de audio e imagen. Este *framework* está desarrollado en C# y orientado a un uso comercial. Además, dispone de numerosos ejemplos de funcionamiento.

Al estudiar la documentación de este *framework* y observar el funcionamiento de los numerosos ejemplos que se ofrecen, decidí crear dos proyectos *".Net"* locales.

El primero fue una API Rest sencilla capaz de recibir una serie de imágenes como entrada, crear un modelo, utilizar dichas imágenes para entrenar el modelo creado, guardar el modelo creado y/o entrenado y guardar las imágenes que se han utilizado.

La API Rest, al ejecutarse en local tenía ciertas limitaciones, como por ejemplo a la hora de guardar las imágenes o los modelos.

El segundo proyecto consta de una aplicación de Microsoft Windows Forms (2018) que simplemente disponía de un campo de entrada de datos y otro de salida.

Desarrollada la API Rest local y un formulario capaz de recibir los datos de entrada y mostrar la salida correspondiente comencé con la labor de configuración y entrenamiento de un modelo.

Al entrenar un modelo y realizar clasificaciones para comprobar el funcionamiento del *framework* llegue a la conclusión de que aparentemente cumplía todos los requisitos necesarios para su uso en el desarrollo del proyecto.

A continuación, informé a Jaime de los resultados obtenidos y decidimos que lo más conveniente sería crear un nuevo proyecto, pero esta vez utilizando Microsoft Azure (2018) para la gestión tanto del servidor como de las imágenes y modelos generados.

Antes de comenzar con este Trabajo de Fin de Grado había desarrollado una API Rest cuyo objetivo era ayudarme a comprender el funcionamiento de C#, lenguaje de programación que conocía, pero con el que no había realizado desarrollos anteriores, por lo que Jaime se encargó de la reutilización de parte del código creado para crear una nueva API Rest alojada en Microsoft Azure (2018) además de utilizar otros servicios para la gestión de las imágenes y de los modelos.

A mediados de 2018, Jaime me informa de la aparición de una nueva librería de Inteligencia Artificial, TensorflowJS (2019) ya que anteriormente contemplamos la posibilidad de utilizar Tensorflow (2019) en su versión Python.

Al ser JavaScript un lenguaje de programación que aprendemos durante el grado de Ingeniería del Software decidí comprobar el funcionamiento de TensorflowJS (2019).

En primer lugar, busqué la documentación y algunos ejemplos para ver si esta nueva librería ofrecía una interfaz más sencilla para los desarrolladores.

Al estar familiarizado con JavaScript, tanto la documentación como los ejemplos observados de TensorflowJS (2019) me pareció posible la creación de una aplicación de prueba para observar su comportamiento.

Antes de comenzar a desarrollar una aplicación de prueba utilizando TensorflowJS (2019) decidimos que independientemente de utilizar TensorflowJS (2019) teníamos que crear un proyecto *"Front-End"* de cara a los usuarios.

Al trabajar ambos con tecnologías como NodeJS (2019)*;* React (2019)*;* Webpack (2019)*;* Typescript (2019) decidimos que, en este proyecto, para el desarrollo de la aplicación *"Front-End"* se utilizarían dichas tecnologías.

Jaime se encargó de crear el proyecto base y crear un repositorio Git que ambos utilizaríamos como control de versiones de dicho código.

Al disponer de dicho proyecto base comencé con la instalación y configuración de TensorflowJS (2019), ya que se diseñó para poder crear modelos y entrenarlos desde el proprio navegador.

Una vez configurado el proyecto "*Front-End*" con TensorflowJS (2019) decidí comprobar el funcionamiento de la librería creando varios modelos para poder entrenarlos más tarde.

A la hora de realizar el entrenamiento del modelo llegué a la conclusión de que las muestras que yo podía obtener con la cámara del teléfono móvil no eran suficientes por lo que decidí buscar plataformas o librerías que me permitiera obtener imágenes rápidamente y en gran cantidad.

Algunas de las plataformas que encontré son: ImageNet (2018); Pexels (2018); Caltech 101 (2018); Caltech 256 (2018).

El siguiente paso fue entrenar un modelo y realizar clasificaciones. En este paso descubrí que al crear una aplicación que se ejecuta en un entorno web nos encontramos con limitaciones de memoria, además de limitaciones con las que nos habíamos encontrado en el pasado a la hora de gestionar los modelos y las imágenes.

Llegados a este punto y con una aplicación web, decidimos reunirnos con Gonzalo para comentar nuestros avances.

En vista de los resultados obtenidos decidimos que habíamos finalizado la búsqueda de librerías o plataformas existentes para la creación y entrenamiento de redes neuronales y podíamos dar comienzo al desarrollo de la aplicación descartando Accord.Net (2018) para utilizar TensorflowJS (2019).

Para solventar el problema de la gestión de los modelos y de limitación de memoria en la aplicación *"Front-End"*, decidimos crear un proyecto *"Back-End"*, una API Rest con NodeJS (2019) y JavaScript, que se conectaría a una base de datos encargada de mantener la información de los modelos y al tratarse de un servidor reducimos los problemas de memoria a la hora de realizar entrenamientos.

Mi labor en este proyecto fue desarrollar la parte *"Back-End"* reutilizando partes de código que previamente había desarrollado para el *"Front-End"* con algunas modificaciones debidas al cambio de entorno. Este proceso se detalla en el apartado dedicado al desarrollo de la aplicación.

2 INTRODUCTION

2.1 PRELIMINARY

In the last few years, with the methodological and technological progress, new and different applications have been developed, whose usefulness in different areas is more than remarkable. Among them, methods developed under the paradigm of deep learning (Deep Learning) and more specifically the so-called Convolutional Neural Networks (CNN) stand out.

In this sense, the applications developed in the world of intelligent transport and as an aid to driving in automated vehicles, currently evolving towards full autonomy, where these same techniques are called to play a certainly relevant role, are well known. In this area, applications aimed at the detection of obstacles, pedestrians, traffic signs on the tracks and the lines delimiting the traffic lanes stand out, as can be seen, for example, in the latest models of the vehicles of major brands such as Mercedes Benz with its PRE-SAFE system (2019), which detects the possible dangers in such a way that it tries to mitigate them or, if this is not possible, to prepare the environment so that the damage caused to the occupants is as least as possible.

It is evident that these kind of applications increase road safety in all areas, with possibilities such as, that intelligent vehicles equipped with these technologies assume, at certain times or continuously, their own decisions. For example, this can be seen in the latest Tesla models with the addition of Tesla Autopilot (2019), which allows the driver to be released from the task of driving, when required, giving the vehicle an almost complete autonomy when driving, being able to follow the established route by itself and detecting all the necessary obstacles to avoid causing damage or breaking the law. This autonomy is almost complete. They force the driver to remain attentive, detecting for example if their hands are holding the steering wheel, in case their intervention is necessary at some point.

These new applications have also led to progress in many areas of safety, not only in terms of road safety; but for example, in the security of the access to the properties as can be vehicles, farms, factories, guarded enclosures, etc. with a simple mobile device, a fingerprint or a video camera capable of detecting the face of users, with the only requirement of accessing remote servers through current communication systems, and in the near future making use of 5G networks which will grant even more quickness to the answers.

Another area of application of these technologies comes from the world of Unmanned Aerial Vehicles (UAV), also known as drones. The UAVs are equipped with various technological devices, highlighting digital color or even multispectral cameras, GPS, pressure sensors, height, temperature… among other possible (Pajares 2015). The combination of all these devices allow someone to obtain information about the environment, process it and make decisions accordingly. The RNCs have much to say in this area, so that the provision of greater intelligence gives them a greater level of autonomy.

It is well known the use of UAVs in electronic commerce, where the delivery orders in the shortest possible time and safely, has become a challenge of great relevance through these vehicles. The logistics and distribution companies are betting strongly on this line. However, with the increase in demand for these companies, problems that previously did not exist, appeared, for example, the delivery of orders in complex geographical areas, in very short

periods of time or in time slots not compatible with the receivers. When making these shipments, as previously indicated, time is essential and directly impacts the business and reputation. Therefore, these companies decide to invest in new technologies that allow shipments to be made quickly and at the lowest possible cost.

The UAVs, within their limitations of autonomy and charge, have the capacity to access remote areas and perform routes through the air without crew, which implies savings in personnel, in times of delivery, in vehicles and fuels and even promotes an improvement for the environment. All this is orientated to achieve the highest level of efficiency in the delivery of orders. In this sense, it is well known that UAVs have an specific route planned, guided by GPS signal, to the point of delivery. Although, there is a major problem just at the time of delivery, which is to determine the exact point where to deposit the object, taking into account that it must be safe. Because in certain circumstances the accuracy of the GPS can be an obstacle, a possible solution is to recognize an area of landing or urban area that ensures the delivery without risk by using recognition techniques of textures in images taken with the camera on board. However, this involves a significant risk, in the sense that these areas may not be identified correctly. Therefore, another solution consists in recognizing a platform, designed specifically for this purpose, so that it is an exclusive object not present in the operating environments. This is precisely where the present work is focused, so that, through the techniques based on the RNC, it is possible to identify it. To increase security in deliveries, it is possible for each order to generate a unique platform design, which avoids ambiguities in the recognition.

It should be noted, moreover, that this approach is not exclusive for the case of logistic deliveries indicated, but it is well known how, on occasions, UAVs must land for battery recharging operations once the period of autonomy has been exhausted. Autonomous vehicles, in the future, must park in specific parking spaces, sometimes in environments where the GPS signal does not reach. Consequently, in these cases the provision of an identification platform for such purposes is of vital importance. The techniques developed in this project are valid also for these specific cases.

Therefore, returning to the objective of this work and given that there are already works on recognition of landing platforms for UAVs (Piñeiro 2016; Arroyo y col., 2017; Garcia-Pulido y col., 2017). It has been determined that a feasible solution to the problem consists in the generation of a simple arrangement of colored geometric figures, which can be printed on a A4 size sheet, which can be done by the receiver of the requested delivery services. In this way, the user places the platform on the ground, providing the geographic coordinates, so that the UAV, upon detecting the platform, would land or deposit the request on it. The main difference regarding the designs presented in the previous references is the use of CNNs.

2.2 MOTIVATION

As previously indicated, the main motivation that originates this work lies in the detected need to develop intelligent methods for the recognition of objects, in this case a landing platform using CNN.
On the other hand, given the boom and the technological progress, it is convenient and necessary to deploy a flexible tool that provides a technological capacity for future developers in this type of applications.

Moreover, situating itself in the point of view of potential users, it is of great importance the fact of having a tool with the capacity of providing a view on the performance of the applied methodologies for the recognition of any type of objects by images, not only platforms, so as to offer and open business possibilities in any field, not only through the use of UAVs.

Finally, indicate that from the point of view of the teaching-learning process, the fact of developing an application of these characteristics involve a relevance challenge with regarding the acquisition of knowledge in the field of cutting-edge technologies in clear boom and progress.

2.3 OBJECTIVES

The general objective of this project is the development of an application for the identification of a landing platform for a UAV by means of captured images with the video camera attached to the vehicle.

In addition to this main objective, the following specific objectives are identified:

1. Perform a review of techniques, algorithms and software already available in the field of Artificial Intelligence and image recognition.

2. Apply Deep Learning techniques, using CNN, for the recognition of the platform.

3. Create a flexible and friendly application that allows carrying out the model management process from its creation and training to its use in the detection of landing platforms for UAVs.

4. Perform experimental tests for the validation of the developments, in order to determine the viability of the model with the greatest possible precision.

As for the work plan, all phases of the project have been carried out weekly, but not in its entirety, so the time devoted to each objective is approximate, therefore, six weeks have been devoted to the first point of this sections and eight weeks to sections two, three and four.

3 REVISIÓN DE METODOLOGÍAS

Como se ha indicado previamente, en el presente trabajo se presenta el desarrollo de una aplicación flexible con una sencilla y amigable interfaz para el usuario, para gestionar modelos de reconocimiento de imágenes mediante técnicas basadas en aprendizaje profundo con RNC.

En la presente sección, se exponen los aspectos relacionados con las metodologías utilizadas, identificando los siguientes aspectos: a) Métodos de reconocimiento de una plataforma de aterrizaje mediante procesamiento de imágenes; b) Descripción general de las RNC; c) Clasificación con imágenes y d) Descripción de TensorflowJS.

3.1 RECONOCIMIENTO DE UNA PLATAFORMA DE ATERRIZAJE MEDIANTE PROCESAMIENTO DE IMÁGENES

García-Pulido y col., 2017 diseñaron un sistema experto que proporciona la orientación del UAV con respecto a la plataforma junto con el grado de confianza en su reconocimiento. La plataforma utilizada se muestra en la Figura 1 cuya patente es la descrita en Cruz y col., 2012. Se trata de una figura geométrica sobre fondo blanco conteniendo una serie de figuras geométricas inexistentes o al menos poco comunes en la naturaleza.

Figura 1: Plataforma de aterrizaje original, binarizada y etiquetada en regiones

Chen y col., 2017 proponen una plataforma de aterrizaje con una figura en blanco y negro que contiene un pentágono en negro situado en el centro de un círculo que sirve como indicador del punto exacto de aterrizaje. El método para su reconocimiento se basa en la técnica conocida como *Faster Regions with Convolutional* (Faster R-CNN), incluyendo las típicas arquitecturas de las RNC. Polvara y col., 2017 también aplicaron CNNs para detectar un círculo negro rodeando una cruz también negra, ambos sobre un fondo blanco. Por lo tanto, en estos dos últimos casos, se trata de planteamientos muy próximos al realizado en este trabajo, si bien con otros modelos de red y sin la flexibilidad otorgada por la herramienta diseñada en el presente trabajo.

Wang y col., 2016 propusieron una figura en forma de H, como la existente en los helipuertos convencionales, encerrada en un círculo, ambas en negro sobre fondo blanco o viceversa. La imagen se segmenta mediante la extracción de lo que se conoce como puntos de interés (esquinas, *corners*). Sharp y col., 2001 también utilizaron cuadrados blancos y negros con detección de *corners* con el mismo propósito. Zhao y Pei, 2013 utilizaron una forma en H de color verde sobre la que aplicaron el descriptor SURF (Speeded-Up Robust Features) de puntos de interés (Bay y col., 2008), que es invariante a pequeñas rotaciones y cambios de escala. Saripalli y col., 2006 también utilizaron una forma H y técnicas de procesamiento de

imágenes, incluyendo filtrado, umbralización, segmentación y etiquetado con determinación del ángulo de orientación entre las regiones etiquetadas con respecto al UAV. Lange y col., 2008, 2009 propusieron una figura similar a la de Saripalli et al., 2012 conteniendo diversos anillos concéntricos blancos.

Cocchioni y col., 2014 diseñaron sendos círculos concéntricos junto con dos triángulos equiláteros más pequeños, todos ellos de diferentes dimensiones. Li y col., 2012 también utilizaron seis círculos concéntricos.

Guili y col., 2009 diseñaron una plataforma con forma de T, cuyo material produce distintos niveles de intensidad en el infra-rojo con respecto al entorno, por lo que su visualización es posible con cámaras operando en el infra-rojo.

Los patrones del tipo AprilTags (Olson, 2011; AprilTag, 2018) fueron utilizadas por Ling, 2014; Kyristsis et al., 2016; Garrido-Jurado et al., 2014. Se utilizaron técnicas de detectores de bordes para el procesamiento de las imágenes. Los marcadores definidos en ArUco Markers, 2018 se utilizaron en Chaves y col., 2015 junto con el filtro de Kalman para guiar al UAV durante la última fase del aterrizaje sobre barcos. Araar y col., 2017 utilizaron también los AprilTags con técnicas de agrupamiento mediante según la magnitud del gradiente. Nguyen et al., 2017 utilizaron tres círculos concéntricos que definen ocho áreas blancas y negras.

En definitiva, tal y como se deduce de los planteamientos anteriores existen trabajos en la línea del presente proyecto, relativos al reconocimiento de una plataforma mediante imágenes digitales, si bien bajo distintos planteamientos y puntos de vista. En efecto, la propuesta planteada en este trabajo proporciona una herramienta abierta y flexible, basada en un modelo de RNC con un diseño específico, que permite el re-entrenamiento de la red con ejemplos propios para el reconocimiento de la plataforma de aterrizaje de VANT u otros objetos de interés con posibilidad de ampliación a distintos ámbitos. La posibilidad de diseños de plataformas específicas abre un amplio abanico de posibilidades en el sentido expresado previamente como elementos motivacionales.

3.2 Breve descripción de las redes convolucionales neuronales

Las RNC son un tipo especializado de redes neuronales (LeCun, 1989), con una topología basada en rejilla para el procesamiento de datos, tales como series temporales o imágenes, rejillas *n*-D, siendo *n* un valor que especifica la dimensión de la rejilla. Esta estructura se conoce como tensor. En la descripción de las RNC siguen las dos referencias básicas siguientes: Goodfellow y col., 2016 y Dumoulin y Visin, 2016.

El término *convolucional* hace referencia a la operación matemática de *convolución*, que es una operación lineal ciertamente especializada. De esta forma puede decirse que las RNC son redes neuronales que utilizan la convolución, en vez de la multiplicación de matrices, en al menos una de sus capas. La operación de convolución se aborda aquí desde el punto de vista de estas redes neuronales, que no se corresponde exactamente con el mismo concepto aplicado en otros ámbitos tales como el procesamiento de señales o desde el concepto matemático puro. Las RNC encajan en los principios neurocientíficos que rigen el aprendizaje profundo.

Por otra parte, en casi todas las redes RNC se aplica una operación denominada *pooling*, que viene a ser una reorganización de píxeles mediante alguna operación de agrupación realizada sobre una ventana en los resultados intermedios que se obtienen tras la convolución.

En esencia, la convolución consiste en el desplazamiento de un núcleo, conteniendo una serie de valores, a lo largo de la imagen, realizando las operaciones de producto y suma que aparecen en la Figura 2, para obtener los valores P y Q y así sucesivamente. En esta imagen, los desplazamientos del núcleo son de una posición a la siguiente, pero en lugar de desplazarse una posición de izquierda a derecha y de arriba abajo, el desplazamiento podría ser de más de una unidad, es lo que se conoce como *stride*.

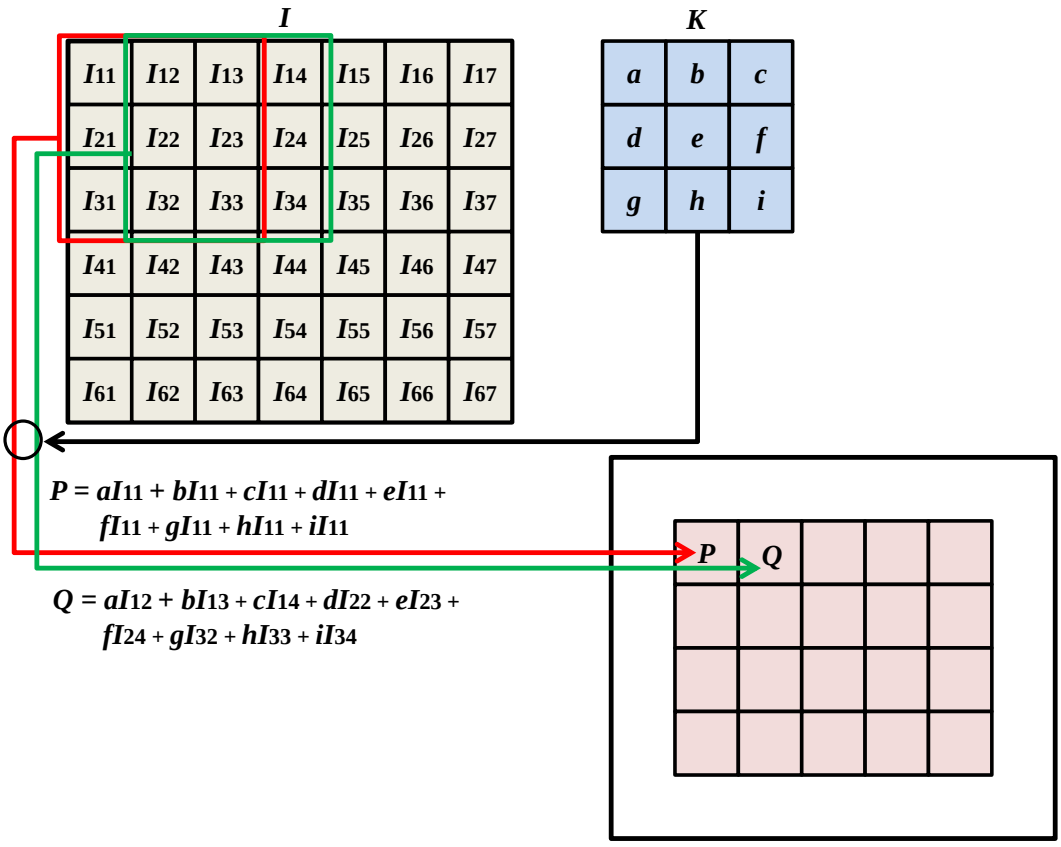

Figura 2: Operación de convolución con desplazamiento de una ventana

En cada localización, se obtiene el producto entre cada elemento del núcleo y el elemento de entrada sobre el que se solapa y los resultados se suman para obtener la salida en la posición actual. El procedimiento se repite utilizando diferentes núcleos para formar tantos mapas de características como se deseen. En la Figura 3 se representa una operación de convolución a partir de tres mapas de características de entrada para obtener cuatro mapas de características de salida, utilizando una colección de núcleos *w* de dimensión 4×3×3×3. Siguiendo la línea superior, el mapa de características 1 se convoluciona con el núcleo $w_{1,1}$, el mapa de características 2, con el $w_{1,2}$ y el mapa de características 3 con el núcleo $w_{1,3}$ de forma que los resultados se suman elemento a elemento para formar el primer mapa de características de salida. El mismo procedimiento se repite para las dos líneas intermedias y la inferior de la figura para formar los mapas de características de salida 2, 3 y 4 respectivamente, de forma que los cuatro mapas de salida se agrupan de forma conjunta.

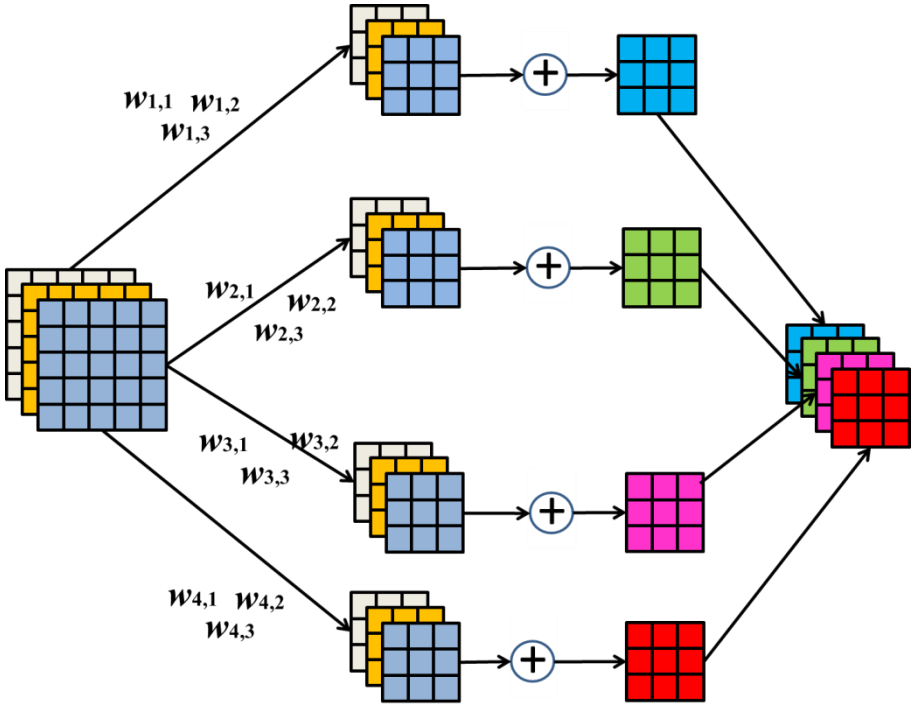

Figura 3: Operación de convolución

Las capas denominadas de *pooling* o agrupaciones proporcionan una importante invarianza a pequeñas traslaciones de la entrada. Existen varias operaciones de este tipo, una de ellas es el máximo (*max*), que consiste en dividir la entrada en ventanas, generalmente sin solapamiento, produciendo como salida el máximo de la ventana. Otra de las operaciones es la media (*average, mean*) de la ventana en la que la salida es el resultado de esta operación sobre la ventana (Boureau y col., 2010a,b; Saxe y col., 2011; Zhou y Chellapa, 1988).

Conviene reseñar que son precisamente los pesos indicados previamente las estructuras que se aprenden durante la fase de entrenamiento, y por tanto son los que ya existen cuando se incorpora un modelo pre-entrenado, siendo éstos mismos pesos los que se ajustan y actualizan durante la fase de re-entrenamiento con los ejemplos proporcionados para este propósito.

3.3 CLASIFICACIONES CON IMÁGENES

Antes de comenzar con la explicación sobre los elementos propios de TensorflowJS (2019), se procede a comentar de forma breve las dos maneras de clasificación de elementos a la hora de realizar una clasificación mediante imágenes:

- **Clasificación multi-clase**: Conocida como clasificación uno de muchos o *One-of-many*, en la cual teniendo un número N de clases disponibles en el sistema, cada entrada pertenece solamente a una de estas clases. A modo de ejemplo, supóngase que un sistema está entrenado para detectar si en una imagen hay un perro, un gato o un coche, y se decide introducir una imagen con un perro por lo tanto el sistema debería ser capaz de indicar que se trata del elemento perro.

En cambio, si se le introduce una imagen que contenga más de uno de estos elementos, indicará solamente uno de ellos; que puede ser, aquel para el cual el resultado de la clasificación esté más próximo a una de las clases con las que se ha entrenado el sistema. La Figura 4 muestra un ejemplo simple e ilustrativo de lo indicado previamente, donde las tres clases definidas en la parte izquierda tienen asignada una terna de valores representada por un cero o un uno, en la parte derecha de dicha figura aparecen imágenes, supuestamente clasificadas con indicación de la clase a la que pertenecen según la terna de valores que las identifica, observándose cómo en la parte inferior aparecen estructuras clasificadas como pertenecientes a una de las tres clases aún a pesar de que la coincidencia con ellas no es total.

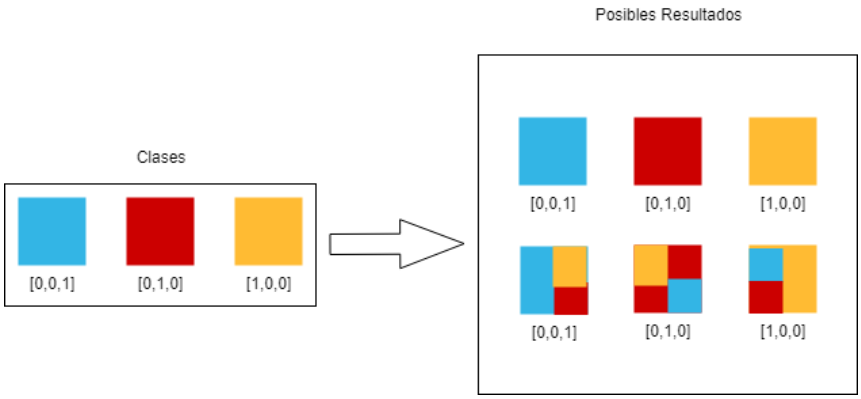

Figura 4: Clasificación multi-clase

- **Clasificación multi-etiqueta**: Siendo N el número de clases disponibles en el sistema, cada ejemplo puede pertenecer a varias clases. Por ejemplo, siguiendo con el mismo ejemplo anterior para detectar un perro, un gato o un coche, al introducir varios de estos elementos en una misma imagen, el sistema debería ser capaz de indicar cuáles de estos elementos aparecen en la imagen y no sólo uno de ellos como ocurría en la clasificación multi-clase. En el ejemplo de la Figura 5 se muestra un ejemplo simple e ilustrativo de este planteamiento.

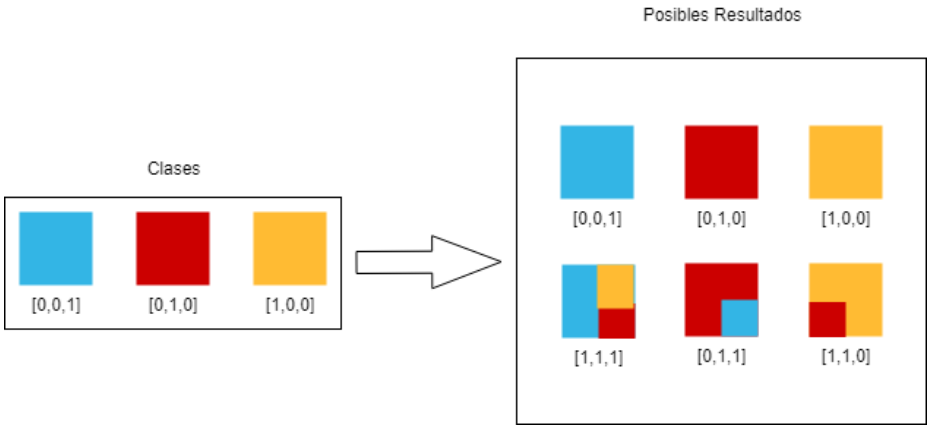

Figura 5: Clasificación multi-etiqueta

3.4 DESCRIPCIÓN DE LOS ELEMENTOS DE TENSORFLOWJS

A continuación, se describen los elementos que componen TensorflowJS (2019) como librería de código abierto para desarrollar modelos de RNC:

- **Tensores**: Son la base de la librería, constituyendo la unidad fundamental para generalizar estructuras escalares, vectores y matrices de N dimensiones. De forma interna, TensorflowJS (2019) representa los tensores como vectores de N dimensiones de tipos básicos de datos, no permitiendo al desarrollador modificar sus valores, por lo que son de sólo lectura, y sólo permite crearlos, utilizarlos y obtener tensores mediante llamadas a los distintos métodos del *framework*. Para modificar los valores de un tensor se utiliza lo que se conoce como *variable*.

- **Variable**: Son tensores que permiten modificaciones a lo largo del tiempo, por tanto, se llaman variables a aquellos tensores que se pueden modificar exclusivamente mediante operaciones.

- **Operación:** Son métodos ya definidos en la librería que permiten realizar cálculos sobre los tensores, de tal manera que pueden realizar sumas, restas, multiplicaciones y divisiones sobre los tensores obteniendo un tensor nuevo de salida que puede ser concatenado con el resultado de otra nueva operación, de forma que todos los tensores intermedios así obtenidos permitan obtener un nuevo tensor resultante.

- **Modelo:** De manera muy simple, se puede definir un modelo como una función de forma que dada una entrada devuelve un resultado. Para ello, un modelo está compuesto de operaciones en su interior. Pero esto sólo ocurre cuando se trabaja a bajo nivel con operaciones sencillas como las comentadas anteriormente. Si se desea trabajar a alto nivel, es posible definir un modelo mediante la API de *layers* o capas de TensorflowJS (2019). Los modelos pueden ser entrenados con muchos datos y se utilizan para realizar predicciones o clasificaciones. TensorflowJS (2019) almacena los modelos en formato JSON de tal manera que permite una sencilla exportación e importación de modelos a través de dicho formato.

- **Capas o *Layers*:** Son una serie de operaciones a alto nivel que abstraen al desarrollador de su implementación a bajo nivel, en este grupo se ubican las capas de convolución de las RNC.

Cuando se trabaja con redes neuronales es importante conocer los elementos requeridos, a saber: las funciones de activación, las funciones de pérdida y las funciones de optimización.

- **Función de activación o *Activation Function*:** En redes neuronales en general, la función de activación permite emular el comportamiento del cerebro a la hora de recibir un estímulo para identificar si ese estímulo corresponde o no a una acción específica. A nivel matemático la operación que realiza una función de activación es traducir los valores de los resultados en ceros y unos dependiendo de la función utilizada. Se dispone de dos tipos de funciones de esta naturaleza: lineales y no lineales.

- ***Función lineal***: Es una función polinómica de primer grado; es decir, una línea recta. No suele ser utilizada con redes neuronales ya que no restringe los valores en ningún intervalo. En la Figura 6 se muestra un ejemplo de este tipo.

$$f(x) = mx + b$$

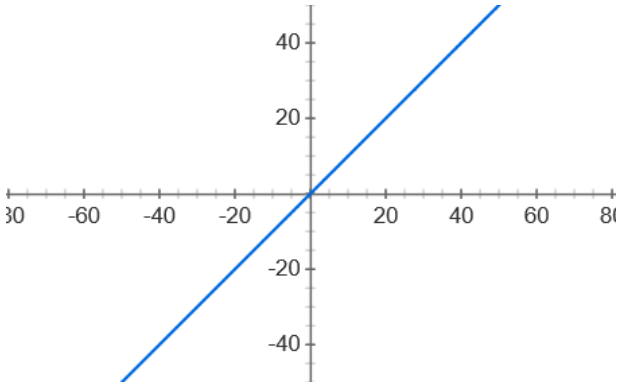

Figura 6: Función lineal

- ***Función no lineal***: Al contrario que las funciones lineales, las no lineales son las más comúnmente utilizadas como funciones de activación. TensorflowJS (2019) utiliza dos tipos característicos como son la sigmoide y el rectificador, aunque bien es cierto que en este último caso se presenta una linealidad a tramos, de ahí su nombre ReLU (*Rectified Linear Unit*). La Figura 7 y la Figura 8 muestran sendas representaciones junto con sus correspondientes funciones matemáticas.

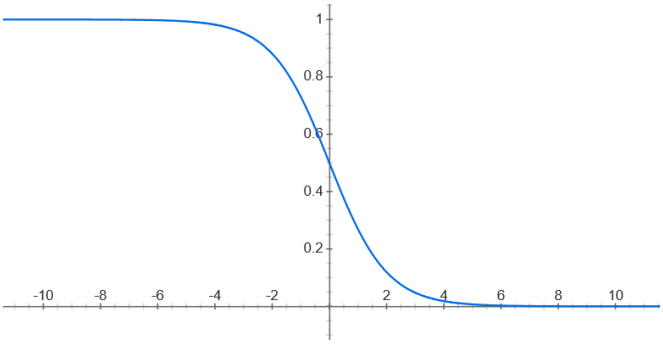

Figura 7: Función sigmoide

$$f(x) = \frac{1}{1 + e^x}$$

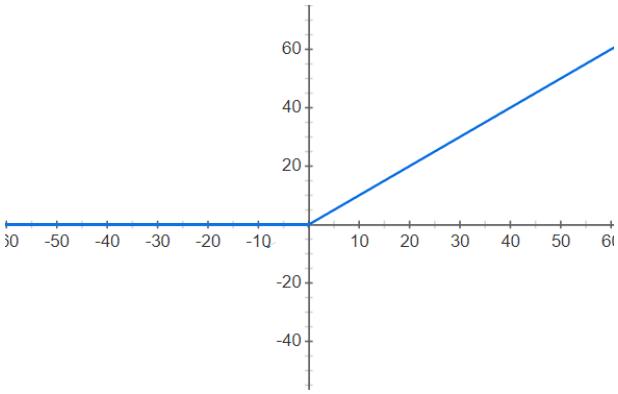

Figura 8: Función Rectificador (ReLU)

$$f(x) = \max(0, x)$$

- **Función pérdida o *Loss Function*:** Es una función que, dada una entrada y un resultado esperado, mide cómo de bueno es el resultado obtenido en cada iteración. Como ejemplo de tal función se puede incluir la función *Cross Entropy,* la cual es muy utilizada en su forma binaria (conocida como *Binary Cross Entropy)* para clasificaciones con sólo dos clases:

$$CE = -\sum_{i}^{C} t_i \log(s_i)$$

donde C el número de clases, t_i la clase actual y s_i la distribución actual.

Otro ejemplo, puede ser la función *Categorical Cross Entropy* conocida también como *Softmax loss* la cual se compone de la función de activación *Softmax* sumada a la función perdida *Cross Entropy*. Esta función se utiliza frecuentemente en clasificaciones con un número de clases superior a dos,

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \quad CE = -\sum_{i}^{C} t_i \log(f(s)_i)$$

donde C el número de clases t_i la clase actual y s_i la distribución actual.

- **Método de optimización:** Utilizado para ajustar los pesos de conexión y por tanto los parámetros de aprendizaje. Se basan generalmente en determinar el error cometido en cada iteración con el fin de minimizarlo a medida que se proporcionan ejemplos a la red, es decir a media que se realiza el correspondiente entrenamiento sobre la misma. Los algoritmos de optimización más utilizados en este tipo de redes son el conocido como descenso de gradiente y Adam el cual es una combinación de las ventajas del algoritmo *Adaptative Gradient Algorithm* (AdaGrad) y el algoritmo *Root Mean Square Propagation (RMSProp)*.

- **Épocas o Epochs:** El número de veces que el modelo procesa el mismo conjunto de datos durante el entrenamiento. Es decir, el número de iteraciones que va a realizar el modelo sobre los datos de entrenamiento para que, en cada iteración, el modelo adquiera un mayor grado de aprendizaje, y por lo tanto un mejor ajuste de los pesos de la red en relación a las entradas que se le proporcionan.

4 ANÁLISIS Y DISEÑO

En este apartado se describen los aspectos relacionados con el análisis y diseño del proyecto, que comprende: a) análisis previo; b) descripción de las tecnologías específicas de desarrollo utilizadas en el ámbito de las RNC; c) planteamiento y definición de las pruebas de concepto; d) Desarrollo de la aplicación; e) gestión del proyecto junto con sus herramientas asociadas. En todos los casos se justifican las elecciones mediante las razones suficientes que avalan las decisiones tomadas en estos ámbitos.

4.1 ANÁLISIS PREVIO

Antes de comenzar con la especificación de requisitos, se plantea el análisis y las ideas previas al desarrollo dentro del equipo, para así poder explicar el planteamiento final de la propuesta formulada.

La mayoría de los trabajos relativos al reconocimiento de plataformas de aterrizaje se fundamentan en la aplicación de técnicas de procesamiento de imágenes con el fin de segmentar la figura o figuras que componen la plataforma dentro de la imagen. Así, el énfasis se pone en la identificación de las figuras, junto con las relaciones geométricas que las componen. Muchas de estas técnicas presentan problemas derivados de la iluminación, ya que las plataformas se ubican en entornos de exterior causando problemas tales como reflejos o atenuación de los colores, e incluso vista parcial de las figuras. Dada la evolución y avances en técnicas que no requieren segmentación, como es el caso de las RNC, se ha optado por este tipo de técnicas para solventar la mencionada problemática. Estas técnicas consideran el objeto de forma global sin tener en cuenta las partes de que se compone, es decir las figuras que forman la plataforma. De hecho, ya se han propuesto técnicas en este sentido, tal y como se ha indicado previamente (Chen y col., 2017; Polvara y col., 2017). Si a esto se añade la potencialidad de las tecnologías web y de comunicación factibles entre el VANT y una supuesta estación base para intercambio de imágenes y mensajes, el conjunto permite definir la estrategia propuesta en el proyecto.

Centrándose en el proyecto, el objetivo consiste en desarrollar una aplicación, que permita a un usuario entrenar modelos con RNC mediante imágenes de objetos identificados, de forma que, una vez finalizado el aprendizaje, se puedan realizar las correspondientes clasificaciones, en este caso la plataforma de aterrizaje del VANT. De esta manera, el propio usuario, realiza el entrenamiento necesario para reentrenar un modelo predefinido que se puede utilizar para tal propósito. Como parte fundamental del desarrollo se propone la realización de diversas pruebas para la validación del modelo.

Un aspecto relevante para considerar no es la definición de la plataforma de ejecución de los algoritmos, ya que se plantea dicha ejecución tanto desde un computador o desde cualquier dispositivo móvil. Por lo tanto, es necesario pensar en un diseño que permita tener siempre centralizada la lógica de la aplicación, con accesibilidad desde cualquier tecnología y diferentes entornos. Esto tiene la ventaja de no tener que repetir código cada vez que se quiere desplegar la aplicación en un entorno diferente.

Adicionalmente a lo mencionado anteriormente, cabe destacar la importancia de abstraerse de la plataforma o sistema operativo a utilizar por el usuario para realizar el entrenamiento o

la clasificación. Es decir, además de no tener que desplegar múltiples veces la misma lógica según el sistema operativo, resulta muy interesante también disponer de una única tecnología, con la que mostrar la interfaz, para desarrollarla sólo una vez y con capacidad de despliegue en múltiples entornos simultáneamente.

Por lo tanto, para platear el primer requisito, se propone crear un servicio web que se despliega en un servidor web, de tal manera que la aplicación sólo disponga de un único entorno de producción donde poder procesar su lógica. También, de esta manera, al disponer de un servicio web, la tecnología a utilizar resulta indiferente; ya que, mientras se tenga la posibilidad de permitir desarrollar una interfaz visual, realizar peticiones HTTP y obtener archivos del sistema para seleccionar imágenes es suficiente. Bajo este supuesto, aún se avanzó más allá planteando una segunda idea, cuya base consiste en abstraerse de la plataforma donde se muestra la interfaz, lo cual se aborda mediante el desarrollo de una tecnología que permita su despliegue inmediato en múltiples plataformas, por lo tanto, se decidió llevar a cabo un desarrollo web con NodeJS (2019), de tal manera que ejecutando una serie de *scripts* se está en disposición de desplegar la aplicación en una servidor web, en un escritorio para Windows, OS o Linux; o en una aplicación móvil ya sea Android o IOs.

De esta manera, se plasma una idea bastante interesante, que permite a los usuarios de la aplicación acceder desde cualquier lugar, en cualquier dispositivo con el único requisito necesario de disponer de conexión a internet y, en caso de no poseer alguno de los sistemas mencionados, de disponer de un navegador web compatible con HTML5. La Figura 9 muestra un esquema donde aparecen los diferentes componentes del diseño. En efecto, se puede observar que el planteamiento consiste en la utilización de un servidor web con NodeJS (2019) instalado y ejecutando la aplicación. Este servidor lanzaría las consultas necesarias contra otro servidor donde estaría localizada la base de datos de la aplicación. Este otro servidor de base de datos no tendría por qué ser un servidor físico distinto al servidor de la aplicación. En el esquema también se muestra la idea de ejecutar la aplicación desde diferentes tipos de dispositivos, en este caso se ha decidido poner una pequeña muestra de los sistemas operativos más comunes hoy en día, los cuales serían capaces de acceder mediante su correspondiente navegador web al servicio o mediante una aplicación nativa del sistema.

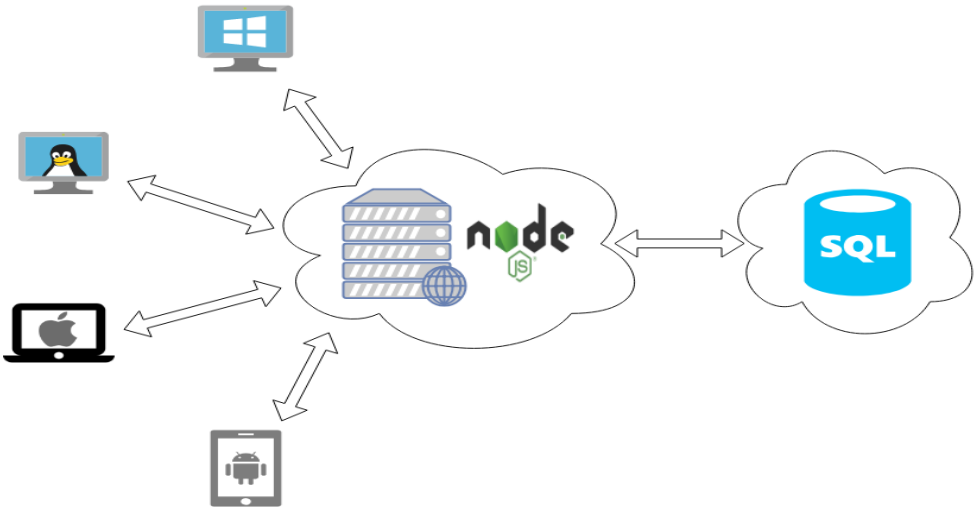

Figura 9: Topología de los componentes de la aplicación

4.2 TECNOLOGÍAS ESPECÍFICAS DE DESARROLLO PARA RNC

Una vez definido el esquema y la topología base de la aplicación, se aborda el estudio de las tecnologías existentes en el ámbito de las RNC para su integración en dicho esquema. Se han estudiado las que se exponen a continuación, de tal manera que en algunos casos se han creado sencillas aplicaciones con el objeto de determinar su viabilidad. El estudio realizado por orden cronológico es el siguiente:

- **AlexNet (2018):** Es un modelo pre-entrenado que utiliza redes neuronales convolucionales. Posee una fácil integración con lenguajes como MATLAB y Python; y, además, permite utilizar la GPU durante la fase de entrenamiento. A pesar de su potencialidad y prestaciones, este modelo fue descartado debido a que el objetivo del diseño era crear una aplicación de escritorio y/o web donde poder desplegar los conceptos metodológicos y conceptuales que la definen. Se fundamenta en el uso de tensores para la definición de la estructura de sus componentes.

- **Tensorflow (2019)**: Es una librería de código abierto desarrollada por el equipo de Google Brain para uso interno, que fue lanzada posteriormente bajo licencia Apache 2.0, para poder crear modelos de redes neuronales, entrenarlos e identificar patrones utilizando tanto la potencia de la CPU como de la GPU. Proporciona además una cierta flexibilidad otorgando control al programador para hacer uso de sus componentes a medida de las necesidades requeridas. Tensorflow (2019) fue inicialmente descartado por el hecho de poseer un *framework* demasiado amplio y complejo, que, además, tenía su propia filosofía para desarrollar los modelos frente a otros con mayor flexibilidad. Como su nombre indica, está basado en el concepto de tensores, como en el caso anterior. Durante la fase de búsqueda de librerías, se determinó que Tensorflow (2019) sólo tenía implementaciones publicadas en Python, C++, Java y Go con ejemplos que no eran del todo fácilmente comprensibles. Debido a todas estas circunstancias, inicialmente se descartó como tecnología a desplegar en la aplicación. En el momento de estudio de esta librería no se conocía la potencialidad de TensorflowJS (2019), que se encontraba en fase Alpha.

- **Microsoft Cognitive Services (2018):** Debido al conocimiento y experiencia previa por parte de uno de los dos miembros del equipo durante el desempeño de su carrera profesional, se decidió desarrollar una aplicación de prueba y concepto utilizando esta tecnología. En el campo de la Visión por Computador permite la creación de un proyecto en la "nube" con Microsoft Custom Vision (2018), que resulta de fácil configuración, permitiendo, además, subir imágenes y asignar una etiqueta a cada una de ellas, entrenar un modelo y realizar clasificaciones manuales o a través de API REST. Este proyecto en un principio parecía interesante ya que permitía la abstracción correspondiente a la parte de aprendizaje profundo, permitiendo así concentrar el esfuerzo en la creación de una aplicación capaz de gestionar y entrenar los diferentes modelos, con capacidad suficiente para reconocer plataformas de aterrizaje para un VANT. Finalmente, se descartó esta opción porque no se consiguió acceso a la documentación clara sobre las técnicas y/o algoritmos utilizados para la obtención del resultado o del funcionamiento de las RNC.

- **Accord.Net (2018):** Es un *framework* específico de aprendizaje automático, combinado con librerías para el procesamiento de audio y de imágenes de forma inteligente. Está totalmente escrito en C#. Durante el transcurso del proyecto, se

desarrolló también una aplicación con Microsoft .Net Framework 4.5 (2018), utilizando C# como lenguaje de programación, en la cual se implementó una interfaz de usuario muy básica utilizando la "API" Microsoft Windows Forms (2018), que es exclusiva para entornos bajo Windows. No obstante, se desconocía que ya se encontraba en estado obsoleto en el momento de su estudio, a pesar de haberle dado todo el protagonismo del desarrollo a interfaces de usuario para Windows con las Microsoft Universal Windows Platforms (2018). El desarrollo realizado permitía introducir imágenes, entrenar una red neuronal, procesar las imágenes y devolver un resultado. No obstante, requerían grandes cantidades de imágenes para realizar un entrenamiento capaz de generar clasificaciones de forma razonable. Además, no hacía uso de técnicas de aprendizaje profundo y la interfaz de usuario aparecía muy limitada y antigua por el uso de Microsoft Windows Forms (2018). Todas estas razones condujeron al descarte de esta opción.

- **Microsoft CNTK (2018):** Es un *framework* de Microsoft para desarrollo de aplicaciones con inteligencia artificial de aprendizaje profundo, muy potente, escrito en .NET Standard lo cual permitía un desarrollo multiplataforma de la aplicación. Tiene desarrollos en C++, C# y en Python. Al igual que Tensorflow (2019), fue descartado por su complejidad y también porque no se podían encontrar suficientes ejemplos de desarrollo ni documentación descriptiva para poder iniciar al menos las primeras pruebas sencillas con el fin de determinar sus posibilidades.

- **TensorflowJS (2019):** Es un *framework* de código abierto desarrollado por Google, pero a diferencia de Tensorflow (2019), éste está completamente escrito en lenguaje JavaScript. Durante el año 2018, fueron liberadas una serie de versiones Alpha de este *framework*, con una alta potencialidad en lo que se refiere a su propia estructura y al lenguaje utilizado. Con el fin de realizar implementaciones con TensorflowJS (2019) mediante JavaScript, Google reescribió todo el Core de Tensorflow (2019) (originariamente escrito en C++) en JavaScript. La principal ventaja que otorgaba este nuevo Core en este lenguaje era la posibilidad de poderse utilizar tanto en proyectos de escritorio multiplataforma con NodeJS (2019) como en un navegador web con soporte de JavaScript. Gracias a esta posibilidad es posible realizar un desarrollo completamente orientado a navegador con prácticamente toda la funcionalidad que puede proporcionar el desarrollo de escritorio. Esto permite abstraer el proyecto del sistema operativo donde debe ser ejecutado. La versión Alpha de TensorflowJS (2019) proporciona una sencilla API (Application Programming Interface) como abstracción a los algoritmos y técnicas necesarias para el entrenamiento de las RNC. El 9 de marzo de 2019 fue lanzada la primera versión estable de este *framework*, por lo que se optó definitivamente por esta opción, reajustando todos los componentes desarrollados hasta el momento.

4.3 Planteamiento y Definición de Pruebas de Concepto

Como se ha comentado previamente durante el análisis de las tecnologías existentes, se han desarrollado pequeñas aplicaciones con sus correspondientes pruebas de concepto para comprender mejor el funcionamiento de dichas librerías. Estos desarrollos se han creado conforme a los estándares establecidos por la Ingeniería del Software e Ingeniería del Conocimiento.

Como apoyo logístico se decidió almacenar el código en repositorios privados de Github (2018) con acceso permitido a los dos miembros del equipo. Durante los primeros desarrollos se decidió centralizar todo en un único repositorio; pero en el desarrollo de la aplicación final se decidió utilizar dos repositorios: uno para la aplicación frontal y otro para el servidor; ya que, no tenía dependencias una aplicación con la otra. El uso de estos repositorios permite al equipo acceder de forma simultánea a las últimas versiones del código para poder seguir realizando modificaciones sobre esta última versión; además de permitir comprobar cada uno de los cambios que se han ido generando.

A continuación, se describen los desarrollos realizados junto con los módulos que los caracterizan, tomando como base las tecnologías específicas de desarrollo para RNC.

A) Accord.Net y Microsoft CNTK

El primer desarrollo del proyecto comenzó con la elección de Accord.Net (2018) como librería de apoyo para desarrollar e implantar el sistema de entrenamiento y clasificación. Con tal propósito se plantearon las dos siguientes fases:

- **Fase 1**: Consistió en un desarrollo completamente local, en el cual se decidió implementar una sencilla interfaz de usuario realizada con Microsoft Windows Forms (2018), evolucionando más adelante hacia Microsoft Windows Forms (2018) y una capa de negocio que exponía una API, también local, con capacidad de publicar métodos que permitían obtener los modelos (en esta fase se decidió denominar “networks” a los modelos), entrenarlos y realizar las pertinentes clasificaciones.

- **Fase 2**: Consistió en la transformación del proyecto anterior en un servicio Web, utilizando Microsoft ASP .Net Web APIs (2018) para implementar de forma sencilla un servicio API REST. Además, se incluyeron los despliegues del servidor en Microsoft Azure App Service (2018) con almacenamiento de imágenes y modelos en Microsoft Azure Blob Storage (2018). En la Figura 10 y la Figura 11 se muestran los diagramas de casos de uso realizados para esta versión del proyecto.

  Como se puede observar, en la Figura 10 están reflejadas todas las operaciones necesarias para poder leer y almacenar imágenes. Para la lectura de imágenes, se contemplaron dos maneras, ya sea mediante la obtención de su URL propia de Microsoft Azure Blob Storage (2018) de tal forma que una web podría referenciarla y mostrarla sin necesidad de descarga o también mediante la descarga directa de la imagen para aquellas aplicaciones no capaces de referenciar la URL directamente.
  Por otra parte, en la Figura 11 se observan las acciones que podría realizar el usuario que tuvieran dependencia de los *networks* o modelos, como son el listado de múltiples formas, la adición, el entrenamiento y la clasificación. En esta versión aún no se había contemplado el borrado de imágenes ni modelos por parte del usuario ya que el proyecto se encontraba en una fase muy inicial y no se había contemplado esa necesidad.

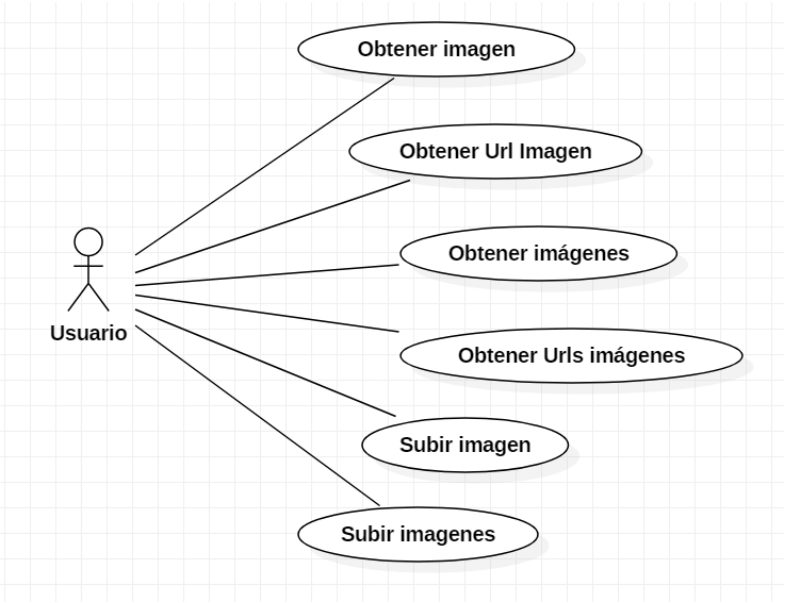

Figura 10: Diagrama de casos de uso de imágenes

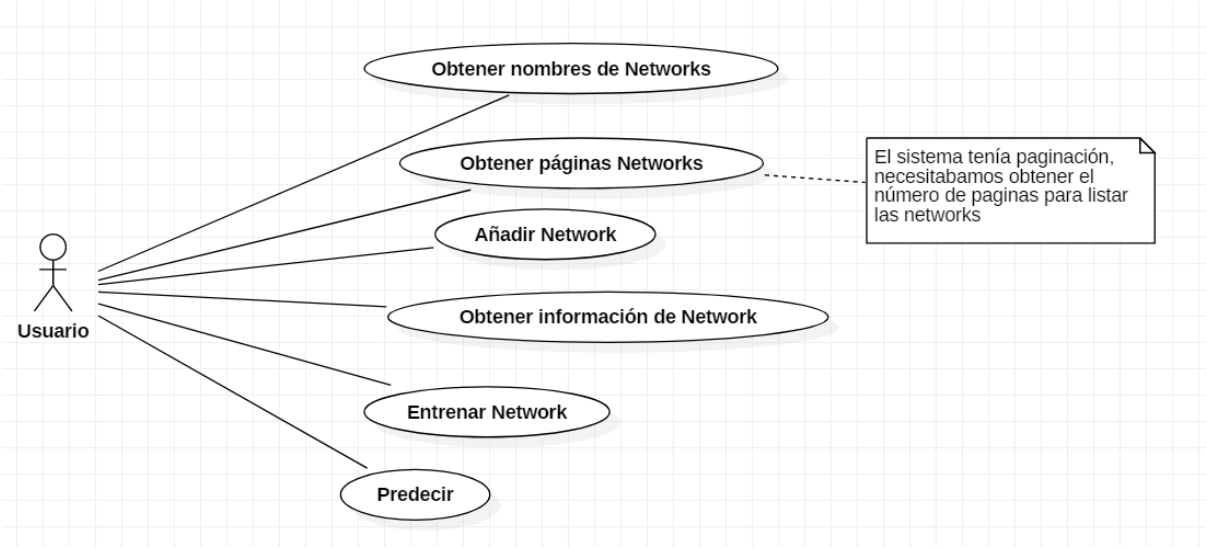

Figura 11: Diagrama de casos de uso de networks o modelos

A pesar de tener un sistema web con una gran funcionalidad, Accord.Net (2018) no proporcionaba los resultados esperados al no cumplir con las expectativas planteadas. La principal razón es su fuerte limitación en relación a los entrenamientos de los *networks* y a las clasificaciones, siendo además necesario realizar los entrenamientos con un elevado número de muestras, en comparación con otras librerías existentes en el ámbito de las RNC para poder clasificar de forma fiable.

Como consecuencia de lo anterior se descartó este planteamiento. No obstante, con el fin de intentar reutilizar parte del trabajo ya realizado con tecnologías Microsoft se intentó reemplazar Accord.Net (2018) por la librería Microsoft CNTK (2018) en el servidor. Si bien, como se ha indicado previamente, la falta de documentación y de ejemplos llevó a descartar esta nueva alternativa por la dificultad que presentaba realizar la aplicación con este nuevo *framework*, originando un giro en el planteamiento del proyecto.

B) TensorflowJS

Una vez comprobado que ni Accord.Net (2018), ni Microsoft CNTK (2018) cumplían con las expectativas planteadas, se optó por utilizar TensorflowJS (2019). Durante el desarrollo del proyecto con TensorflowJS (2019) se consideraron las siguientes tres fases:

- **Fase 1**: Se realizó un primer desarrollo completamente orientado al navegador, sin servidor, para la realización de pruebas con diferentes tipos de capas y parámetros hasta conseguir la configuración deseada para el desarrollo del proyecto, momento que dio paso a la siguiente fase.

- **Fase 2**: Consistente básicamente en una migración, ya que se decidió utilizar un servidor web con una API REST implementada con Express (2019) para realizar los entrenamientos y clasificaciones. En esta fase se desplegaron todos los módulos de la primera versión que tenían dependencia con TensorflowJS (2019) de tal manera que se eliminó toda clase de dependencia en la aplicación frontal con el propio TensorflowJS (2019).

- **Fase 3**: Es la continuación de la fase anterior en la cual se decidió implementar las versiones de los modelos en servidor y también se mejoró y se añadieron más elementos a la interfaz de usuario. Se podría definir esta fase como de madurez en el desarrollo del proyecto.

4.4 DESARROLLO DE LA APLICACIÓN

Para llevar a cabo los objetivos del proyecto, y a partir de la fase 2 reseñada previamente, se plantea el desarrollo de la aplicación en dos partes diferenciadas que evolucionan de forma simultánea, a saber: "*Front-end"* y "*Back-end"*.

Para desarrollar tanto el "*Front-end"* como el "B*ack-end"* del proyecto, se ha decidido utilizar como entorno de desarrollo integrado (IDE) Microsoft Visual Studio Code (2019). Al tratarse de proyectos realizados en JavaScript o Typescript, el equipo consideró que era la herramienta más cómoda y útil para este desarrollo por los siguientes motivos:

- Integración con el terminal o terminales del sistema operativo.
- Multiplataforma.
- Gestión y uso de extensiones.
- Destinado para el desarrollo web.
- Conocimiento previo de la herramienta por parte de los desarrolladores.
- Muy ligera.
- Integración con Git.
- Depuración web en el propio IDE.
- Intellisense, el cual ayuda a los desarrolladores a escribir más rápido.
- Gratuita.
- Open source.

Existen otras herramientas en el mercado como Microsoft Visual Studio 2017 (2018) ahora disponible en su versión 2019, o Netbeans (2019) las cuales, también podían haber servido

para el mismo propósito, pero no se consideran herramientas tan enfocadas para el desarrollo en JavaScript como Microsoft Visual Studio Code (2019).

Microsoft Visual Studio Code (2019) es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Se ha desarrollado para ser utilizado principalmente en desarrollos con *JavaScript*, Typescript (2019) y NodeJS (2019), si bien, gracias a sus numerosas extensiones se puede utilizar para desarrollos en lenguajes como C++, C#, *Java*, *Python*, *PHP*, *Go* o *Unity* entre otros.

Para el almacenamiento y control del código se decidió utilizar Git (2018) sobre repositorios privados de Github (2018). Se decidió utilizar estas herramientas debido a que los miembros del equipo ya habían trabajado con ellas anteriormente, por ello, el uso de estas herramientas garantizaba una mayor fiabilidad y seguridad al equipo.

Cada uno de los proyectos disponía de su propio repositorio privado de Github (2018); ya que, un proyecto era totalmente independiente del otro. Posteriormente, se creó un repositorio que unificaba ambos el cual estaba destinado exclusivamente, a desplegar la aplicación en Microsoft Azure (2018) de tal manera que, al ejecutar el proyecto de "*Back-end"* en el servidor de Microsoft Azure (2018), y posteriormente acceder a la URL, automáticamente se mostrase el proyecto de "*Front-end"* funcionando y conectado al de "B*ack-end"*.

En cualquier caso, la idea era seguir diferenciando ambos proyectos en sus repositorios de manera independiente; ya que, al final son aplicaciones distintas y al unificar el proyecto de *"Back-end* con el de "*Front-end"* se estaría provocando un acoplamiento entre la lógica del servidor y la vista. Esto provocaría que, si en un futuro se decidiera utilizar otro tipo de vista para la aplicación, con otra tecnología, habría que tener en cuenta la actual.

4.4.1 Desarrollo front-end

A continuación, se describe el desarrollo del "*Front-end"* (frontal) de la aplicación con la descripción de los componentes y estructuras que lo componen.

A) TECNOLOGÍAS UTILIZADAS

Se decidió utilizar Typescript (2019) frente a JavaScript por tratarse de un lenguaje tipado, lo que facilita la legibilidad y mantenimiento del código. Si bien, el mayor inconveniente estriba en que todos los *frameworks* utilizados deberían tener su correspondiente librería de tipado para su uso con Typescript (2019), lo que finalmente se hizo.

A continuación, cada uno de los *frameworks* utilizados en el proyecto front-end, dividiéndose en dos tipos:

a) *Frameworks de desarrollo*: Son tales que no afectan de manera directa al proyecto propiamente dicho, sino al programador durante el desarrollo, por lo que su desaparición no provocará cambios en la aplicación. En esta categoría cabe distinguir a su vez:

- **Nodemon (2019)**: Es una utilidad que cuando se ejecuta, permite controlar cualquier cambio realizado al guardar un módulo, de tal manera que reinicia el servicio y aplica los cambios de forma casi inmediata.

- **Electron (2019)**: Permite ejecutar una aplicación realizada mediante HTML y JavaScript como una aplicación de escritorio.

- **Webpack (2019)**: Es un agrupador de código JavaScript, de forma que utilizado junto con la extensión "awesome-typescript-loader (2019)" permite, mediante un archivo de configuración, traducir el código Typescript (2019) en código JavaScript, incrustarlo en un único módulo y colocar este módulo en el lugar deseado del proyecto. Se trata de uno de los *frameworks* más importantes de este proyecto.

- **PM2 (2019)**: Es un administrador de procesos que permite ejecutar aplicaciones JavaScript en segundo plano mediante un *Daemon*. Ha sido utilizado para realizar el despliegue de la aplicación en un servidor web en Microsoft Azure Virtual Machines (2019) sobre una maquina con sistema operativo CentOs (2019).

b) Frameworks del proyecto: Necesarios para el funcionamiento de la aplicación desarrollada con el proyecto, distinguiendo:

- **React (2019)**: Desarrollado por Facebook, es el *framework* más importante de este proyecto y uno de los más utilizados hoy en día junto a Angular 6 (2019) de Google o VUE (2019), debido a que otorga la posibilidad de realizar de manera sencilla un proyecto que sigue el patrón Modelo Vista Controlador (MVC) mediante el desarrollo de componentes y jerarquías de componentes. Para realizar un desarrollo con React (2019), todos los módulos del proyecto que forman parte de una vista deben heredar de la clase "Component" de React (2019), de tal manera que permita transformar automáticamente una clase normal en un componente que tiene un estado y unas propiedades a definir:

  - **Estado:** Son los atributos propios del componente. Para poder modificar su valor, se tiene que realizar una llamada al método "*setState*" heredado de "*Component*"; de tal manera, que React (2019) conoce que ese componente se ha modificado.

  - **Propiedades o "props"**: Son métodos, variables u objetos a definir en el componente que le indican al llamador qué elementos necesita el componente para ser llamado.

  Todos los componentes tienen un método público denominado "*render"* que devuelve código HTML, provocando que cuando algún componente llame a otro, muestre aquello que devuelve el método *render*, pudiendo variar según el estado que tenga el componente o las propiedades que le haya inyectado el llamador. Además, se disponen de otros métodos heredados de "Component" que permiten controlar aún más el estado de los componentes, siendo los más relevantes: "*ComponentDidMount"*, "*ComponentDidUpdate"* y "*ComponentWillMount"*.

Para poder incluir código HTML en un archivo de JavaScript es necesario utilizar archivos con extensión ".jsx" en vez de ".js", si bien en el caso de Typescript (2019) se utilizan archivos con extensión ".tsx" en vez de ".ts".

- **Bootstrap (2019)**: Es un *framework* open source utilizado en proyectos con HTML y JavaScript que permite de forma sencilla construir vistas que se comportan de forma apropiada tanto en dispositivos móviles como en equipos de sobremesa o portátiles.

B) ESTRUCTURA DEL PROYECTO DE FRONT-END

Para la organización de este proyecto se ha seguido una estructura jerárquica de componentes, con diferenciación de dos partes respecto de la aplicación, a saber: rutas y vistas.

1) **Rutas**: Son aquellos componentes encargados de implementar y manejar las URLs de la aplicación al ser accedidas. Cada una de estas URLs renderizan un determinado componente, identificado como *Route*. En la Figura 12 se muestra el árbol de rutas, de forma que en el último nivel del árbol se muestran los componentes de tipo *Route*.

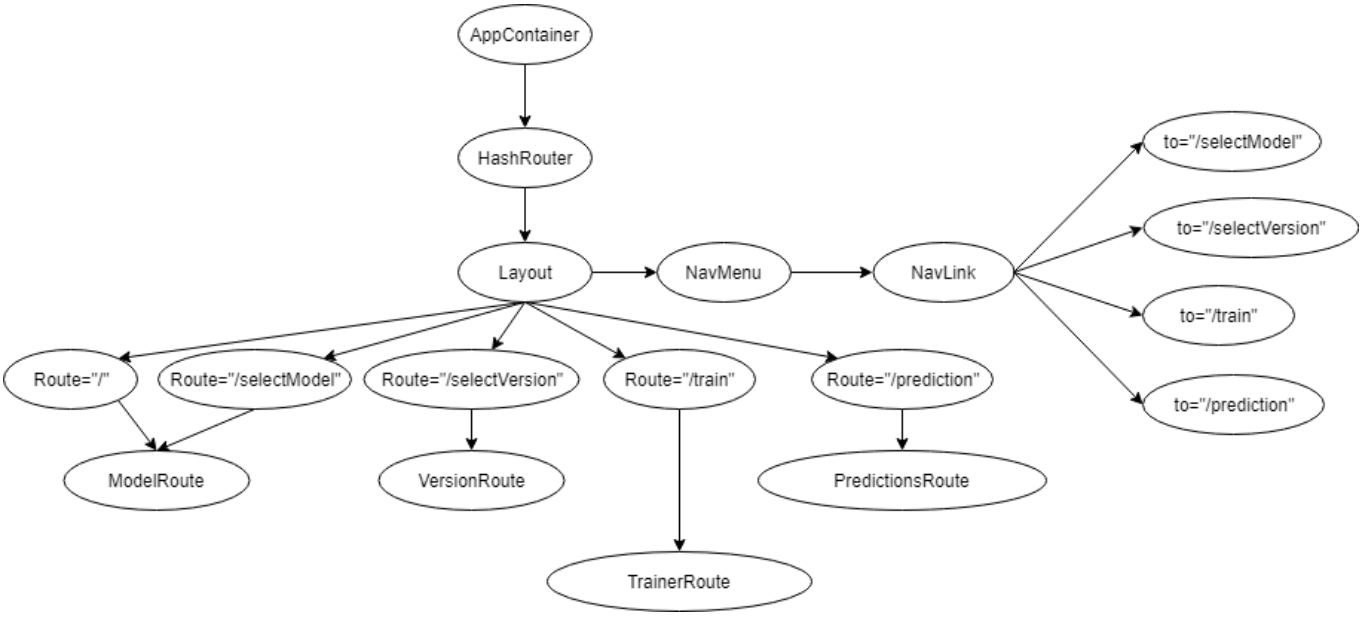

Figura 12: Árbol de componentes de rutas

En el árbol anterior se puede observar cómo han sido organizados los componentes correspondientes, cuya definición es la siguiente:

- **AppContainer**: Componente propio de React (2019), el cual es utilizado para ser establecido como contenedor de la aplicación. Sobre este componente se generan el resto.

- **HashRouter:** Componente encargado de almacenar una porción de la URL, añade un '#' a la URL y todo lo que sigue a partir del '#' lo interpreta como una ruta de la aplicación.

- **Layout:** Componente que se *renderiza* en todas las rutas, incluye la barra de navegación mediante la llamada al componente "*NavMenu*" y almacena los componentes a *renderizar* cuando se hace una modificación de la URL. "*HashRouter"* obtiene las URLs de este componente.

- **NavMenu**: Componente encargado de *renderizar* la barra de navegación de la aplicación junto a sus botones.

- **NavLink**: Componente propio de "*React-Router*", el cual es una librería propia de React (2019) que, se utiliza para crear rutas de acceso en una SPA (*Single Page Application*), que contiene los botones encargados de la navegación por la aplicación. Cambian la URL del navegador.

- **Componentes Route**: Todos los componentes accesibles mediante *clic* a los botones de la barra de navegación ("*NavLink"*). Se corresponde con el componente padre de las vistas de la aplicación.

2) **Vistas**: Son aquellos subcomponentes que forman parte de cada componente Route. Se encargan de mostrar la vista y darle la funcionalidad a la aplicación. Están divididas según la ruta a la que afecten. La aplicación desarrollada contiene cuatro rutas:

- **Ruta 1.- Seleccionar categoría**: Se encarga de mostrar las categorías o modelos existentes, permite seleccionar una categoría de modelo o borrarla. Además, añade la opción de crear una categoría. La Figura 13, la Figura 14 y la Figura 15 muestran respectivamente la vista de selección de categoría, la vista de selección de categoría con una seleccionada y la vista de creación de categoría.

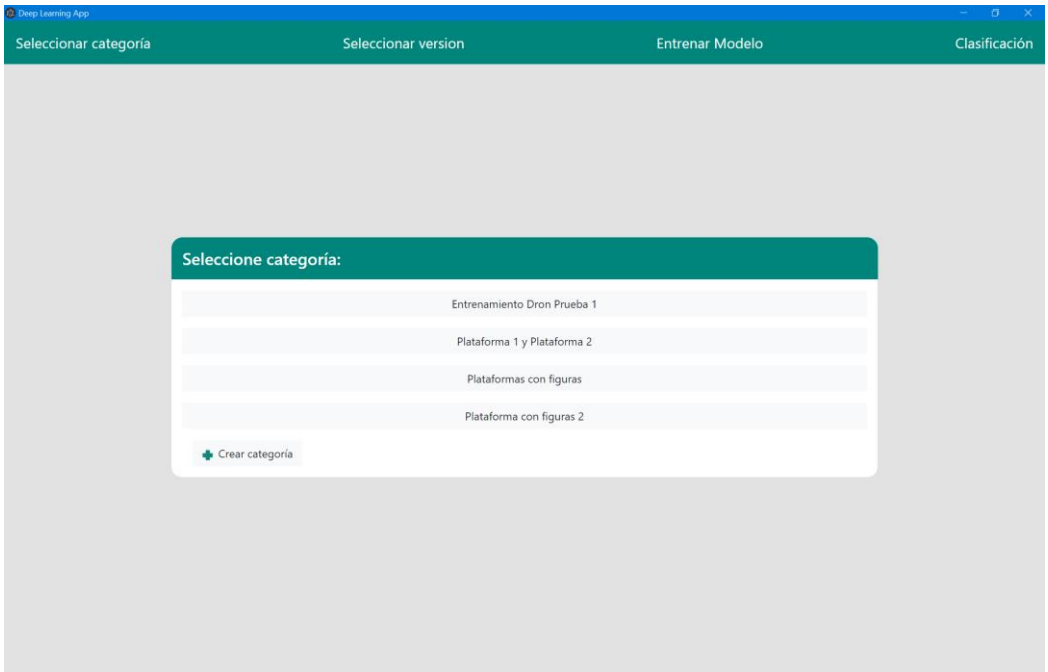

Figura 13: Vista de selección de categoría

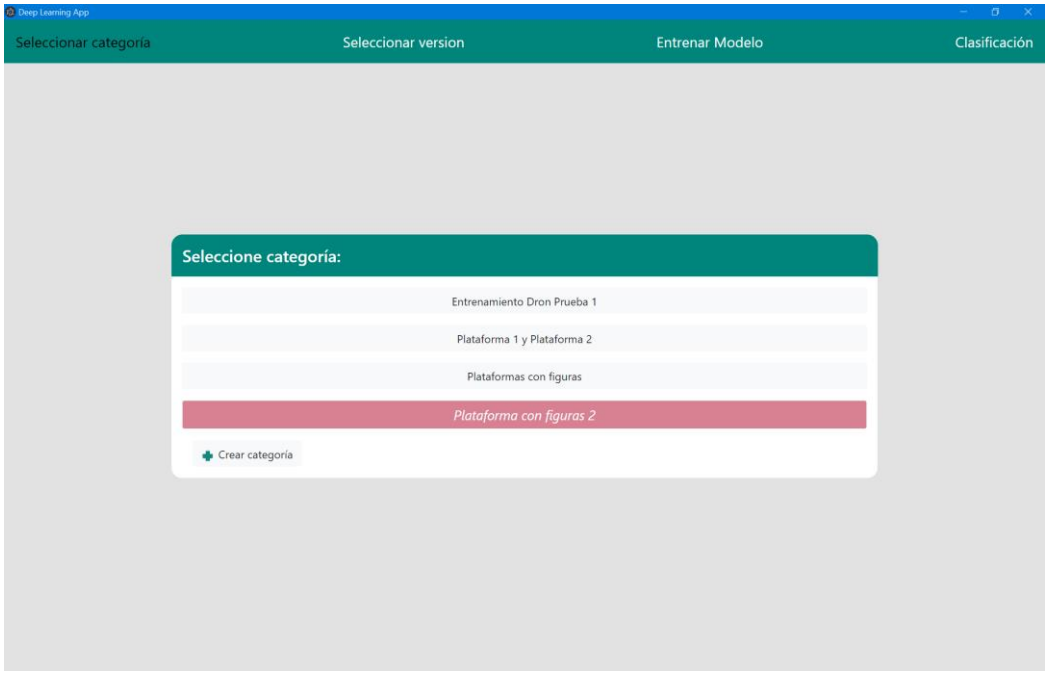

Figura 14: Vista de selección de categoría con categoría seleccionada

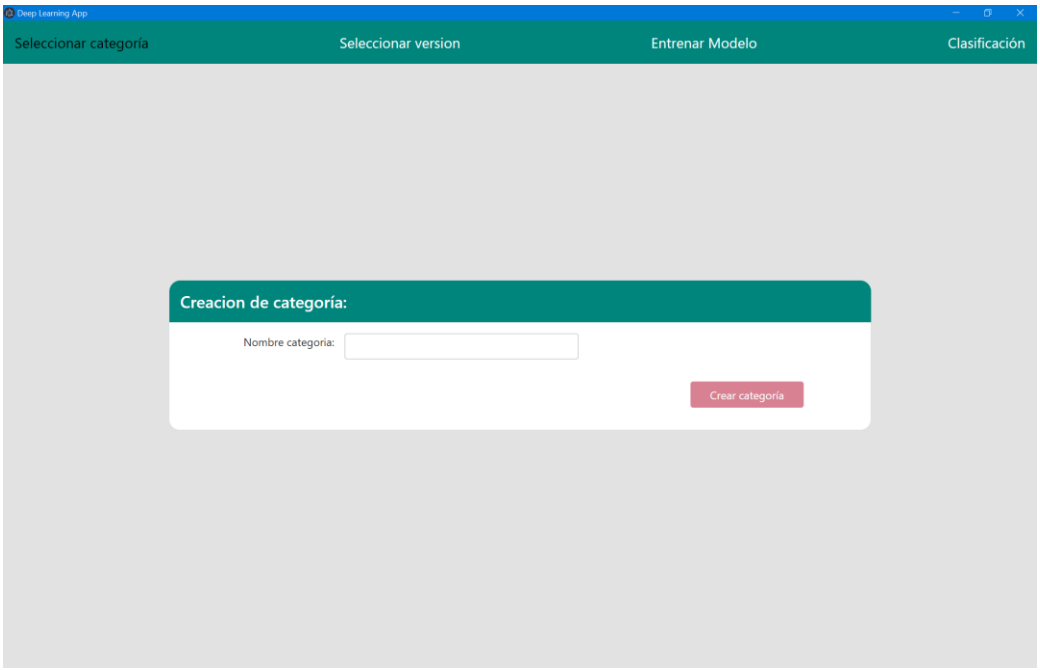

Figura 15: Vista de creación de categoría

- **Ruta 2.- Seleccionar versión**: Se encarga de mostrar las versiones de la categoría del modelo seleccionado, permite seleccionar una versión o borrarla. Además, añade la opción de crear una versión. Como se puede observar, a partir de la Figura 16 y la Figura 17, las vistas de selección de modelo y versión son completamente idénticas, esto es debido a que se están *renderizando* a través del mismo componente, concretamente "*ItemSelector*".

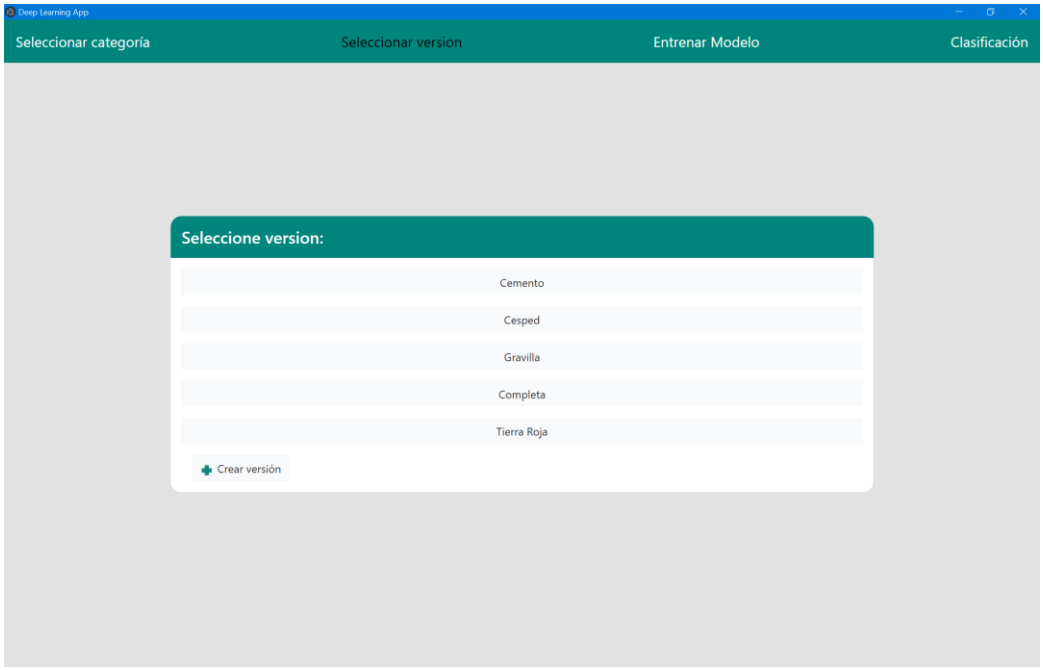

Figura 16: Vista de selección de versión

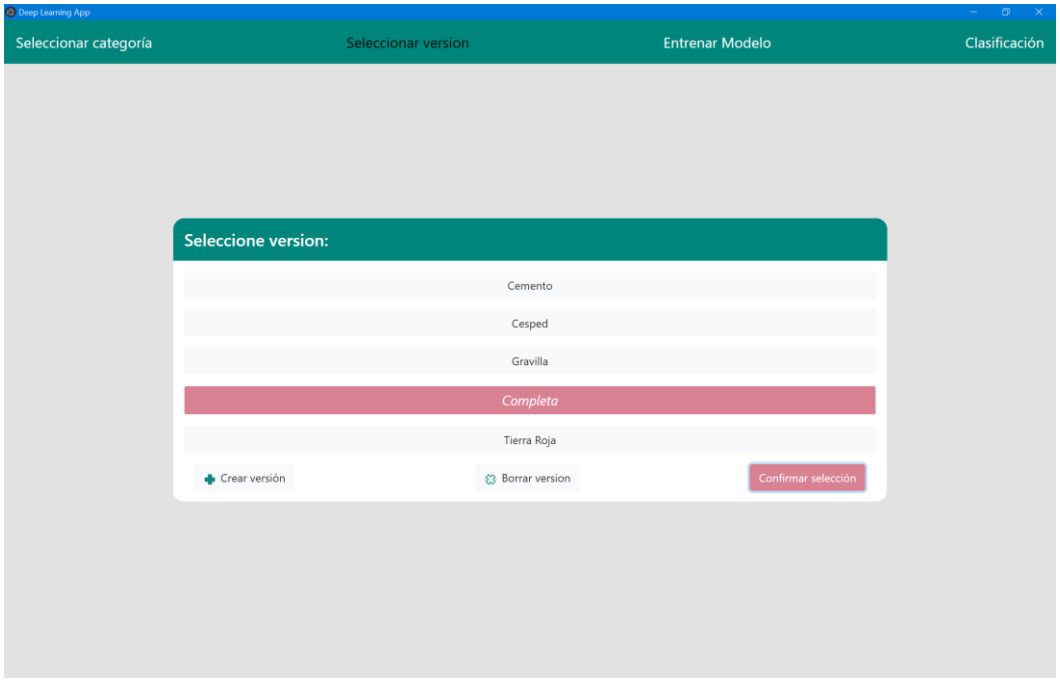

Figura 17: Vista de selección de versión con versión seleccionada

La vista de creación de versiones ofrece un generador de campos de texto dinámico para introducir las clases deseadas al modelo. Este generador lo *renderiza* el componente "*DynamicClassInputs*". La Figura 18 muestra una vista de creación de versión, de forma que al pulsar sobre añadir clase se crea un campo nuevo con una etiqueta indicando el número de clase correspondiente. Al pulsar sobre borrar clase se elimina la última creada, este botón sólo se muestra si existe algún campo de texto generado.

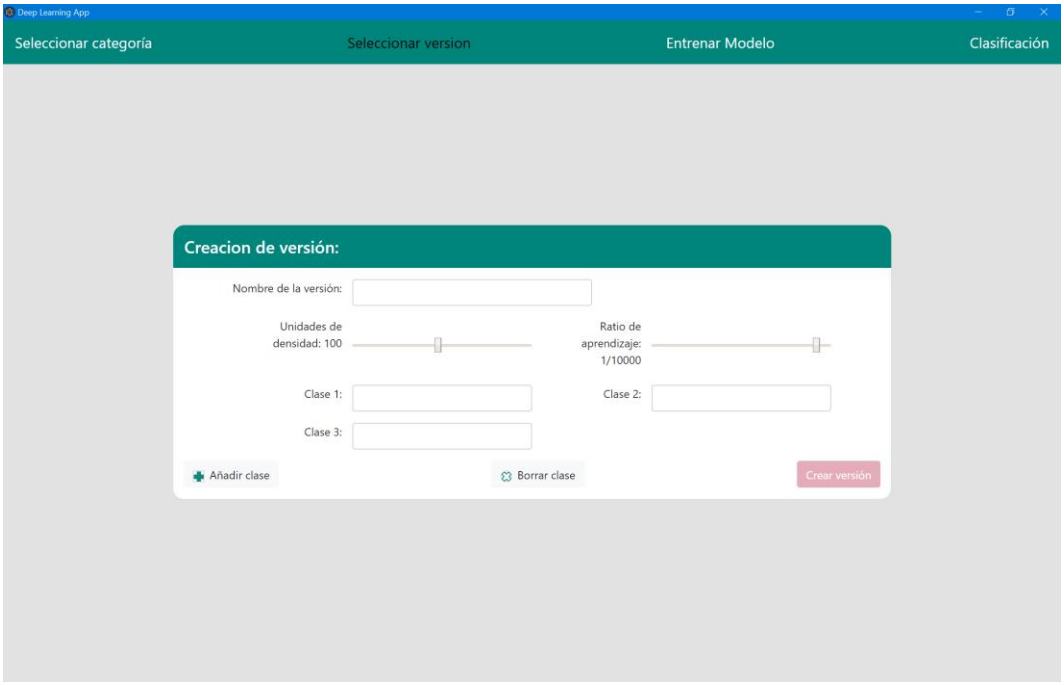

Figura 18: Vista de creación de versión

- **Ruta 3.- Entrenar modelo**: Se encarga de mostrar la interfaz de entrenamiento. Siguiendo la Figura 19, a su izquierda se encuentra un pequeño menú donde aparecen las clases, y los valores por defecto de la versión, este menú es *renderizado* por el componente "*TrainerLeftMenu"*.

  Para añadir imágenes es necesario arrastrarlas sobre el fondo gris, de esto se encarga el componente "*TrainerImageSelector"* el cual, además de mostrar las imágenes, es el encargado de los eventos de selección y marcado de las imágenes. Sobre este componente aparece el componente "*TrainerTopMenu"* que incluye tres opciones según se especifica de izquierda a derecha: borrar imágenes seleccionadas, seleccionar todas las que no estén marcadas y deseleccionar todas. Todos estos componentes son hijos de "*TrainerMainView"*.

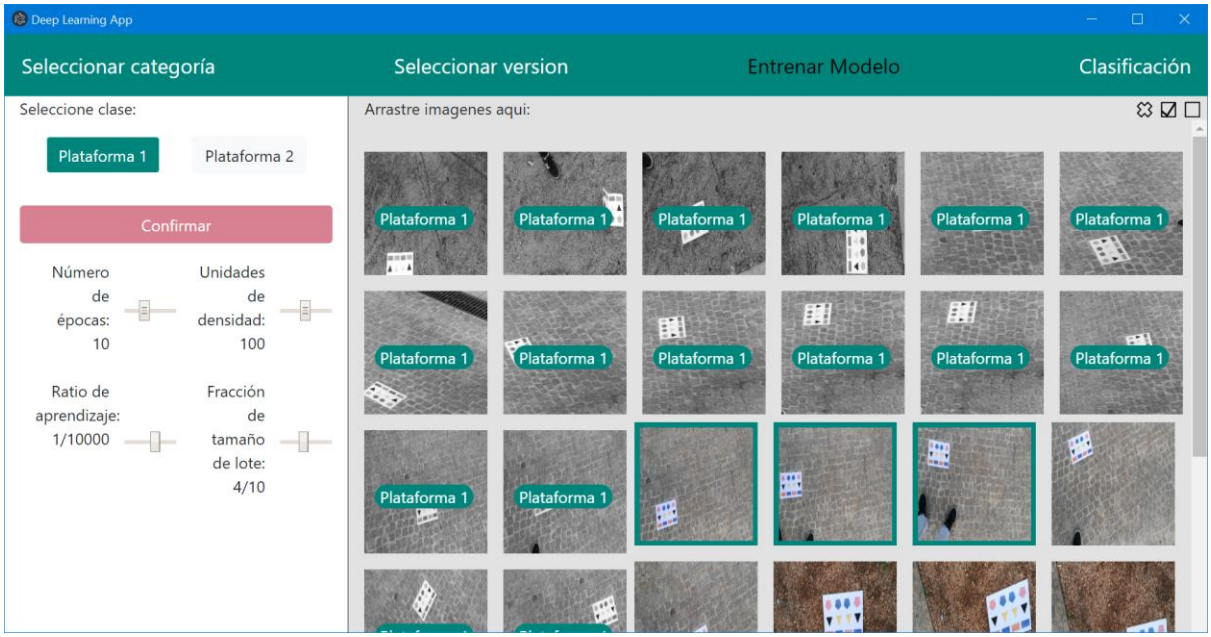

Figura 19: Vista de entrenamiento del modelo

- **Ruta 4.- Clasificación**: Encargada de mostrar la interfaz de clasificación. Es la ruta más simple de la aplicación, al estar sólo compuesta por los componentes: "*ClassificationRouter"* y "*ClassificationMainView"*, Figura 20.

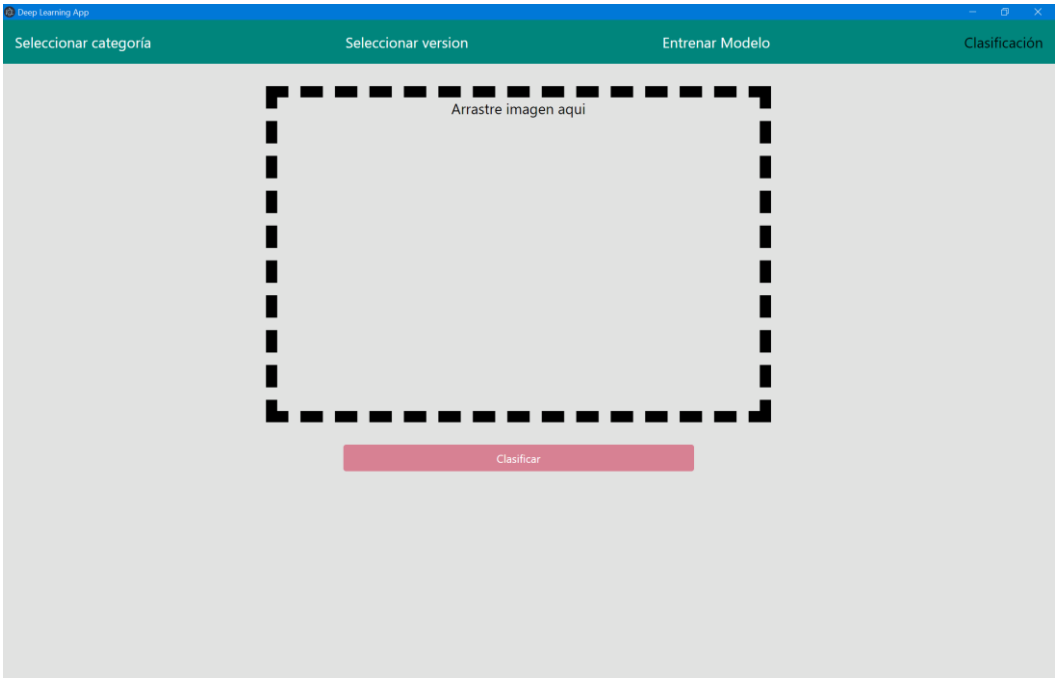

Figura 20: Vista de predicción

Como se puede observar, todas las rutas siguen una misma estructura, ya que, todas tienen su componente "*Route"* y su componente "*MainView"*. Además, si es necesario, añaden los componentes que necesiten para poder implementar el resto de las funcionalidades de la

ruta, de tal manera que la aplicación tiene siempre la misma estructura, resultando así muy sencillo añadir nuevas rutas o funcionalidades.

Es importante destacar, que todos y cada uno de los componentes desarrollados para este proyecto extienden de la clase "React.*Component*" estableciendo el tipo de los "*props*" y el "*state*" mediante el uso de dos interfaces, una para las propiedades y otra para el estado como se muestra en la Figura 21, reemplazando los genéricos que ofrece la clase "React.*Component*" para este fin por las interfaces desarrolladas. En caso de que el estado o las propiedades no contengan atributos ni métodos estas interfaces seguirán existiendo, pero no contendrán ningún tipo de implementación en su interior, esto es para seguir manteniendo la misma estructura en todos los componentes.

```
interface ITrainerLeftMenuProps {
    classesWithSelection: Array<SelectableItem<ClassItem>>;
    onConfirmedClass: (modelClass: ClassItem) => void;
    trainParameters: TrainParameters;
    onEpochsValueChange: (event: React.ChangeEvent<HTMLInputElement>) => void;
    onDenseUnitsValueChange: (event: React.ChangeEvent<HTMLInputElement>) => void;
    onLearningRateValueChange: (event: React.ChangeEvent<HTMLInputElement>) => void;
    onBatchSizeFractionValueChange: (event: React.ChangeEvent<HTMLInputElement>) => void;
    onTrainBtnClicked: (event: React.MouseEvent<HTMLButtonElement, MouseEvent>) => void;
    hideTrainBtn: boolean;
    loading: boolean;
}
interface ITrainerLeftMenuState {
    selectedClass: ClassItem;
}


export class TrainerLeftMenu extends React.Component<ITrainerLeftMenuProps, ITrainerLeftMenuState>
```

Figura 21: Ejemplo de interfaces de los componentes

Como se puede observar en la Figura 21 la clase "TrainerLeftMenu" encargada de mostrar el menú lateral izquierdo de la vista del entrenamiento, hereda de "React.Component" y establece sus "props" al tipo "ITrainerLeftMenuProps" y su "state" al tipo "ITrainerLeftMenuState".

4.4.2 Desarrollo back-end

Para permitir centralizar la lógica del presente proyecto en modo "B*ack-end"* se ha desarrollado una API Rest en un entorno NodeJS (2019). A continuación, se describe el desarrollo del "*Back-end"* de la aplicación con descripción de los componentes y estructuras de que consta.

A) TECNOLOGÍAS UTILIZADAS

Las tecnologías utilizadas para este proyecto son las siguientes:

- NodeJS (2019): Es un entorno de ejecución de JavaScript orientado a eventos asíncronos. Está diseñado para construir aplicaciones escalables. Ofrece un modelo de concurrencia particular respecto a los modelos de concurrencia más comunes existentes hoy en día que utilizan hilos del Sistema Operativo, siendo en la mayoría de los casos insuficientes. Por otra parte, ofrece un entorno concurrente

sin la necesidad de la gestión de los diferentes hilos ya que no realiza directamente operaciones de lectura/escritura.

- Microsoft Azure (2018): Es un conjunto de servicios en la nube orientados a desarrollos profesionales para empresas. En este proyecto se ha utilizado una base de datos Microsoft SQL Server (2018) con licencia institucional de la Universidad Complutense.

La estructura de la base de datos se ha creado con la idea de categorizar los diferentes modelos y no como ampliación de un modelo ya existente, de esta forma, los usuarios tienen que asignar un nombre genérico a un modelo, por ejemplo, “Perros y Gatos” y después crear *versiones* (que son modelos en sí) más específicas, por ejemplo, “Perros negros y Gatos blancos” o “Pastor Alemán y Pitbull”.

La estructura de dicha base de datos relacional se muestra en la Figura 22. Como se puede observar, la base de datos consta de dos tablas, *Model* que se encarga de guardar el nombre genérico del modelo y *Version* encargada de guardar la información de los diferentes modelos creados dentro de esa categoría.

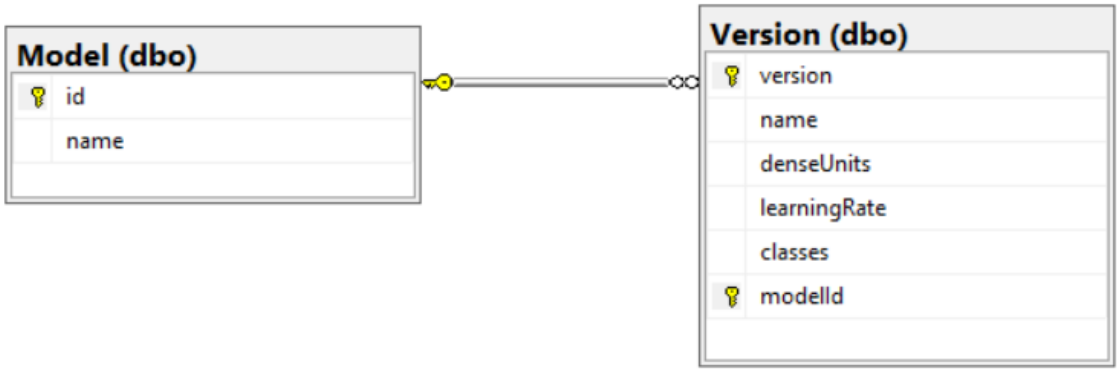

Figura 22: Estructura base de datos

B) IMPLEMENTACIÓN API REST

Para el desarrollo de la API se han utilizado diferentes tecnologías y librerías disponibles. Como se ha comentado anteriormente, al utilizar un entorno NodeJS (2019) se dispone de gran variedad de librerías que se pueden instalar fácilmente mediante el gestor de paquetes npm (2019), mediante el comando: ‘npm -i {nombrePaquete} {parámetros opcionales de npm (2019), normalmente van precedidos de ‘--’}’.

Las librerías fundamentales para el desarrollo que se plantea son:

- **Express (2019):** Es un *framework* que permite crear una infraestructura web rápida y flexible en un entorno NodeJS (2019). Es fundamental en el desarrollo de la API ya que se utiliza para crear toda la infraestructura del proyecto.

- **File System (2019)**: Es una librería que facilita las operaciones de lectura/escritura de archivos del sistema. Se utiliza en la implementación de la API a la hora de guardar las imágenes recibidas para el entrenamiento de los modelos.

- **Multer (2019)**: Es un *middleware* que sirve para gestionar la carga de archivos, en este caso imágenes al servidor.

- **Sharp (2019)**: Es una librería que permite convertir imágenes de gran tamaño y diferentes formatos en imágenes más pequeñas y ligeras. Esta librería es fundamental a la hora de entrenar un modelo ya que las imágenes recibidas podrían tener cualquier formato y tamaño, por lo que el entrenamiento del modelo podría fallar o en el peor de los casos funcionar, pero alterando la consistencia del modelo.

  Para evitar crear modelos "corruptos" se ha tomado la decisión de redimensionar todas las imágenes recibidas, antes de realizar un entrenamiento a la dimensión de 224×224×3 en formato *JPEG*. Cabe mencionar que la aplicación admite dos tipos de formatos de ficheros de imagen, *JPEG* y *PNG*, si bien a la hora de entrenar el modelo, por razones técnicas que se explican más adelante, solo se utiliza el primero.

- **Canvas (2019)**: Es una librería que normalmente se utiliza en desarrollos "*Front-End"* para crear, cargar y modificar elementos gráficos. Se ha utilizado una adaptación de esta librería para "*Back-end"* por ser una dependencia de TensorflowJS (2019).

- **Pixel (2019)**: Es una librería que permite obtener la imagen como un objeto JavaScript. Esta librería se utiliza junto a Canvas (2019) para extraer la información de la imagen y crear el correspondiente tensor para entrenar un modelo o realizar una clasificación.

- **TensorflowJS (2019)**: Como se ha comentado previamente es una librería que ofrece un conjunto de técnicas, algoritmos y objetos para crear modelos, entrenarlos y poder realizar predicciones sobre ellos, utilizando redes neuronales.

- **MobileNet (2019)**: Es un modelo pre-entrenado para mil clases diferentes y recomendado para utilizar en desarrollos de clasificación de imágenes.

C) CREAR UN MODELO

Como se ha mencionado anteriormente, crear un modelo consiste en realizar una llamada a la API Rest utilizando la ruta "/models/createModel". Es una llamada de tipo *post* que crea la categoría de ese modelo. Para ello se utilizan librerías estándar de NodeJS (2019). Como requisito fundamental para llevar a cabo esta acción se solicita al usuario introducir un nombre para dicha categoría (modelo) que no exista previamente en la base de datos.

D) CREAR UNA VERSIÓN

Crear una versión de un modelo, realmente consiste en crear un modelo dentro de una categoría. El proceso de creación de una versión es el siguiente:

- Desde el proyecto front-end se produce la interacción con el usuario que solicita la creación de una versión. Dicha interacción realiza una llamada a la ruta "/models/createVersion" que solicita los siguientes parámetros:

- *Nombre de la versión*: Representa el nombre del modelo dentro de la categoría previamente creada.

- *Unidades de densidad*: Representa el número de neuronas que se quiere asignar a cada capa, varía en el rango [10,200].

- *Ratio de aprendizaje*: Determina la velocidad con la que se realiza el ajuste de los pesos de la red varía en el rango [10^{-4}, 10^{-1}] durante el aprendizaje de la red, y en consecuencia durante el proceso de minimización del error.

- *Clases*: Son las etiquetas a utilizar a la hora de identificar las imágenes, tanto para entrenar el modelo como para realizar las clasificaciones. Para continuar con dicho proceso se requieren al menos dos clases.

Una vez introducidos y confirmados los datos por el usuario se realiza una llamada al servidor, procediendo con la creación de la versión solicitada.

El proceso de creación de la versión en el servidor se realiza en varias etapas, que se sintetizan como sigue:

1. Carga del modelo pre-entrenado (*MobileNet*). Para crear un modelo que sea capaz de realizar clasificaciones más precisas con pocas muestras se ha utilizado el modelo "*MobileNet"*. Cuando se realice un entrenamiento o una clasificación, las muestras introducidas no serán directamente la entrada al modelo creado ya que si éste fuera el caso se necesitaría gran volumen de muestras, en cambio, estas muestras introducidas serán procesadas e introducidas como entrada al modelo de "*MobileNet"* que va a realizar una clasificación para cada una de las muestras. El resultado de la clasificación de "*MobileNet"* va a ser la entrada al modelo creado, reaprovechando de esta forma la información de la red neuronal de "*MobileNet"*.

2. Configuración del modelo que se está creando. Para ello se utiliza por un lado parámetros del modelo pre-entrenado, como por ejemplo el tamaño de la matriz de datos, es decir, la dimensión de las imágenes de entrada y por otro lado parámetros introducidos por el usuario, como por ejemplo el número de clases, el número de neuronas en cada capa, tipo de función de activación y tipo de modelo.

3. Compila del modelo creado utilizando el ratio de aprendizaje especificada por el usuario.

4. Si la compilación ha tenido éxito se almacenan en la base de datos los datos de la versión dentro de su correspondiente categoría.

5. Dentro del servidor se guardan dos archivos de configuración utilizando herramientas proporcionadas por TensorflowJS (2019) en la categoría de la versión correspondiente.

E) ENTRENAMIENTO

Una vez creado un modelo y una versión, se está en posición de entrenar dicha versión, para ello se requieren al menos tres imágenes, que puede proporcionar el usuario.

Al igual que en las operaciones de creación de modelo y versión, para realizar el entrenamiento de un modelo se necesita configurar ciertos parámetros además de disponer de las muestras necesarias. Para llevar a cabo esta acción se realiza una llamada de tipo "POST" a la ruta "/models/train" especificando:

- El número de épocas o *epochs*.
- Fracción del tamaño del lote o *batch size fraction*.
- Ratio de aprendizaje o *learning rate*.
- Unidades de densidad o *dense units*.
- Las imágenes asociadas a una clase.

Una vez el usuario introduce todos los datos y los confirma, se procede a entrenar el modelo.

En esta fase de entrenamiento se comprueba que cada imagen tenga la resolución requerida de 224×224×3 píxeles. Si el tamaño es superior la imagen se convierte automáticamente a dicho tamaño y se guarda en el servidor.

Una vez transformada la imagen al tamaño necesario, se convierte dicha imagen a un tensor y se procede con el entrenamiento del modelo en sí que consta de las cuatro fases siguientes:

1. Se utiliza el tensor creado con la imagen para utilizar el modelo pre-entrenado y realizar una primera clasificación y validación.
2. Una vez recibida la respuesta del modelo pre-entrenado, se asocia dicha respuesta con la etiqueta introducida por el usuario.
3. Se entrena el modelo TensorflowJS (2019) elegido por el usuario utilizando como dantos de entrada la respuesta recibida (un tensor) por el modelo pre-entrenado y la etiqueta introducida por el usuario, que internamente se convierte a un tensor de una dimensión, que constituye el mapa de características.
4. Concluido el entrenamiento se sobrescriben los dos ficheros de configuración del modelo que acaba de ser entrenado, ya que los pesos de las neuronas han cambiado, esto es se han ajustado y por tanto se ha realizado el aprendizaje.

F) CLASIFICACIÓN

Concluidas las etapas anteriores el usuario puede realizar clasificaciones para comprobar la precisión del modelo aprendido o proceder al suministro de más muestras para un nuevo reentrenamiento con más muestras.

Para que la clasificación sea fiable, la imagen introducida debe tener relación con el modelo; ya que, en caso contrario, el resultado obtenido será incongruente siendo asociada a la clase con la que obtenga mayor similitud. Por ejemplo, si se tiene un modelo que reconoce perros y gatos y dicho modelo ha sido entrenado con una importante cantidad de muestras de gatos blancos y perros negros, al introducir la imagen de un caballo blanco, el modelo indicará que es un gato blanco, ya que no tiene ejemplos de caballo blanco.

A nivel técnico, para realizar una clasificación se procede de la misma forma que durante el entrenamiento. Esto es, la imagen debe tener un tamaño mayor o igual que 224×224×3 píxeles. Dichos píxeles se extraen para crear un elemento *canvas* que utiliza para crear un tensor, de forma que se pueda realizar una clasificación utilizando el modelo pre-entrenado.

Finalmente, el resultado obtenido es un tensor de ceros y unos que se contrasta con la información previamente almacenada en la base de datos para encontrar el nombre de la etiqueta que le corresponde.

G) GUARDAR Y CARGAR UN MODELO

Como se ha mencionado anteriormente, a la hora de crear una versión o de entrenar un modelo, se producen cambios en la red neuronal del modelo creado, el modelo pre-entrenado permanecerá constante, por tanto, para no perder el conocimiento adquirido hay que guardar dos ficheros de configuración:

- El primer fichero, “model.json” contiene toda la información relativa al modelo, incluyendo: tamaño que han de tener los tensores, número de clases, tipo de función de activación, número de capas, tipo de cada capa y referencias a los pesos de las neuronas.

- El segundo fichero, “weights.bin” contiene los pesos asociados con las correspondientes capas.

A la hora de realizar una clasificación, dado que se pueden tener varios modelos almacenados, es necesario restablecer tanto la configuración del modelo requerido como la información relativa a las distintas capas de la red.

TensorflowJS (2019) ofrece funciones asíncronas para guardar y cargar los modelos en función del entorno en el que se ejecute, ya sea descargando estos dos ficheros si se trata de un navegador o guardarlos directamente en local o en remoto si se trata de un entorno NodeJS (2019).

4.5 GESTIÓN DEL PROYECTO

En la definición de los objetivos y la organización del trabajo se menciona la metodología ágil SCRUM para la gestión del proyecto, que se caracteriza por ser utilizada en equipos autogestionados, multifuncionales y trabajo en iteraciones. Además, esta metodología especifica unos roles concretos dentro del equipo, en función de las habilidades de cada integrante. A continuación, se describen los roles utilizados en SCRUM junto con las etapas de adaptación al proyecto:

- **Product Owner**: Caracterizado por una única persona. Representa a los interesados y se encarga de maximizar el valor del producto. Su principal labor es realizar correctamente las historias de usuario, incorporarlas al *product backlog* y priorizarlas de forma regular. Rol ejercido por el profesor ejerciendo labores de guía del proyecto.

- **Scrum Master**: Se encarga de que todo el equipo esté utilizando correctamente la metodología, además de ayudar al *product owner* a gestionar el "Product Backlog" y de estar pendiente de los objetivos establecidos, así como de las fechas de entrega. Rol ejercido por los dos miembros del equipo.

- **Equipo de desarrollo**: Lo recomendable es que el equipo de desarrollo tenga un número de integrantes comprendido entre tres y nueve. Se encarga de convertir el Product Backlog (lista de tareas creada en fases anteriores, con la ayuda del cliente) en iteraciones funcionales del producto. No hay jerarquía dentro del equipo, todos desempeñan el papel de desarrollador. Rol ejercido por los dos miembros del proyecto.

Tras la explicación de los roles, a continuación, se describen las acciones metodológicas aplicadas en este proyecto:

- **Sprint**: El desarrollo del software se realiza mediante iteraciones, también conocidas como "*sprints"* que pueden durar un máximo de treinta días. Cada *sprint* se considera como un periodo de desarrollo que culmina con una pequeña demostración de los cambios y novedades que le son mostrados al *product owner. El objetivo consiste en poder comprobar* el estado del proyecto y si el desarrollo realizado satisface sus necesidades. Para llevar a cabo un sprint es necesario definir el Sprint Backlog, que está formado por historias de usuario.

- **Product backlog:** Es el conjunto de acciones (historias de usuario), que debe realizar el producto final. Deben ser escritas y priorizadas por el *product owner* junto al cliente y con ayuda del *scrum master*.

- **Historias de usuario:** Acciones del producto que se añade al *product backlog*. Tienen la siguiente forma: "Como {ROL de usuario} quiero {Acción a realizar}". Lo importante es especificar el rol y la acción a realizar, indicando la parte del programa donde se realiza la acción definida para el rol.

- **Tareas:** Conjunto de acciones en las que se divide una historia de usuario. El equipo debe desarrollar estas tareas.

- **Sprint backlog:** Conjunto de tareas a realizar durante el sprint, que se introducen en el *sprint backlog*. Permiten la posibilidad de realizar un seguimiento de la ejecución del proyecto y previsiones de avance.

- **Daily Scrum:** Son las reuniones diarias que realizan los miembros del equipo de desarrollo para comentar que los trabajos realizados el día anterior, qué ocurrió, y qué pretenden realizar hoy. Su duración es corta, no debe sobrepasar los quince minutos y se suelen hacer de pie para forzar a que su duración sea breve.

En el presente proyecto no se ha podido aplicar íntegramente la metodología SCRUM por las siguientes razones:

- Complejidad para definir las historias de usuario en el ámbito de las RNC.
- Decisiones de diseño cambiantes en función de las metodologías estudiadas.
- Número reducido de componentes del equipo.
- Limitación de los *sprints* en función de la disponibilidad de los integrantes del equipo.
- Necesidad de realizar reuniones telemáticas.

Para llevar a cabo la gestión de las tareas, se ha utilizado la herramienta Trello (2018).

5 ANÁLISIS DE RESULTADOS

Tras el desarrollo del proyecto se procede al análisis de resultados. Para ello se definen en primer lugar el tipo de imágenes utilizadas tanto para el entrenamiento como para la clasificación. En el resto de la sección se proporcionan los resultados obtenidos durante la fase de clasificación para diferentes configuraciones.

5.1 IMÁGENES

Se han diseñado dos tipos de plataforma, ambas conteniendo las mismas figuras geométricas, si bien con diferentes colores en algunas de tales figuras. Se identifican como: *Plataforma 1* y *Plataforma 2*. En ambos casos situadas sobre diferentes bases: tierra roja, tierra, césped, madera y gravilla. De acuerdo a estas bases, se han creado distintas versiones de RNC que las caracterizan, que son tal y como se identifican en el resto de la sección.

De la Figura 23 a la Figura 32, muestran 10 ejemplos ilustrativos de ambos tipos de plataformas sobre las bases indicadas, que son exactamente las imágenes utilizadas para la fase de clasificación. Para la fase de entrenamiento se han utilizado imágenes de la misma naturaleza, pero todas ellas distintas como es habitual en los sistemas basados en aprendizaje. En la Tabla 1 se indica el número de imágenes utilizadas para el entrenamiento según las versiones correspondientes.

Tabla 1: Número de imágenes de entrenamiento según versión y tipo de plataforma

Versión	Plataforma 1	Plataforma 2
Tierra Roja	13	13
Césped	44	44
Gravilla	40	40
Totales	97	97

Cabe mencionar que las diferentes imágenes tanto para la fase de entrenamiento como de clasificación se han tomado a diferentes alturas y ángulos con respecto a la posición de la plataforma. El objetivo es simular de algún modo las condiciones de vista posibles desde un VANT, si bien de forma limitada al no disponer de un vehículo de tal naturaleza. El rango de altura varía desde 1 a 10m y los ángulos de inclinación entre 0º y 45º con respecto a un eje perpendicular a la propia plataforma. Con esta variabilidad de disposiciones se pretende que la RNC capture distintas características definitorias de las plataformas bajo distintas configuraciones.

En cada una de las versiones creadas se utilizan diferentes parámetros de configuración para la fase de entrenamiento, tal y como se indica convenientemente.

Conviene también reseñar que en el diseño actual sólo se utilizan dos clases, correspondientes a Plataforma 1 y Plataforma 2, por lo que, en el supuesto de suministrar una imagen sin plataforma, ésta se identifica como perteneciente a una de las dos clases indicadas en función del mayor grado de similitud a una de ellas, siendo el resultado en cualquier caso incorrecto.

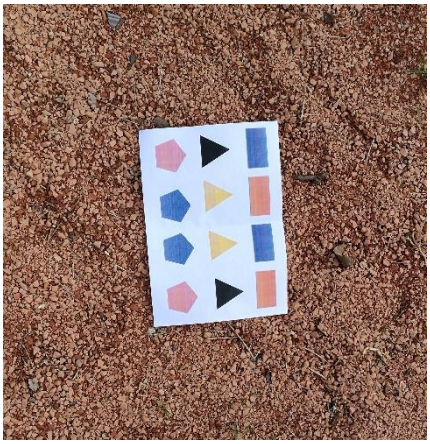

Figura 23: Plataforma 1 Tierra Roja

Figura 24: Plataforma 1 Césped

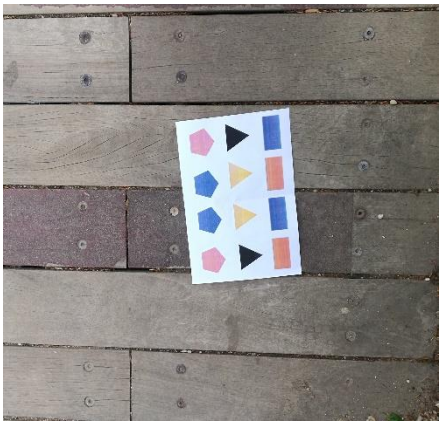

Figura 25: Plataforma 1 Madera

Figura 26: Plataforma 1 Gravilla

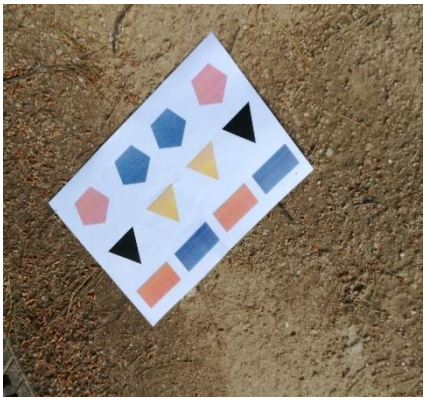

Figura 27: Plataforma 1 Tierra

Figura 28: Plataforma 2 Tierra Roja

Figura 29: Plataforma 2 Césped

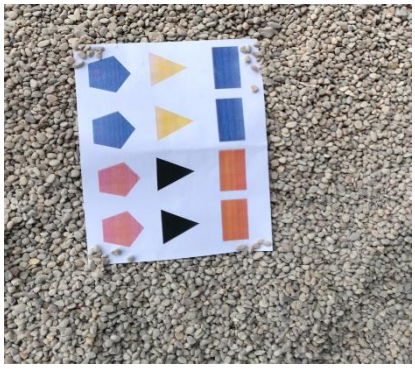

Figura 30: Plataforma 2 Gravilla

Figura 31: Plataforma 2 Tierra

Figura 32: Plataforma 2 Madera

A continuación, se analizan los resultados sobre las versiones “Tierra Roja”, “Césped” y “Gravilla” mientras que la de “Tierra” y “Madera” solo se tendrán en cuenta en la versión completa.

5.2 VERSIÓN SOBRE TIERRA ROJA

Las imágenes utilizadas para el entrenamiento de esta versión son del tipo definido por la Figura 23 y la Figura 28. Tal y como se indica en la Tabla 1, se han utilizado 13 imágenes para la “Plataforma 1” y otras 13 para la “Plataforma 2”.

Los parámetros utilizados para el entrenamiento son los siguientes:

- Épocas: 10
- Ratio de aprendizaje: 10^{-4}
- Unidades de densidad: 100
- Fracción de tamaño de lote: 4/10

Los resultados de la clasificación obtenidos para cada una de las imágenes mostradas en la Figura 23 hasta la Figura 32 se muestran en la Tabla 2 y la Figura 33.

Tabla 2: Resultados de la clasificación para versión sobre tierra roja

FIGURA	RESULTADO
Figura 23: Plataforma 1 Tierra Roja	Correcto
Figura 24: Plataforma 1 Césped	Incorrecto
Figura 25: Plataforma 1 Madera	Correcto
Figura 26: Plataforma 1 Gravilla	Correcto
Figura 27: Plataforma 1 Tierra	Correcto
Figura 28: Plataforma 2 Tierra Roja	Correcto
Figura 29: Plataforma 2 Césped	Correcto
Figura 30: Plataforma 2 Gravilla	Incorrecto
Figura 31: Plataforma 2 Tierra	Incorrecto
Figura 32: Plataforma 2 Madera	Incorrecto

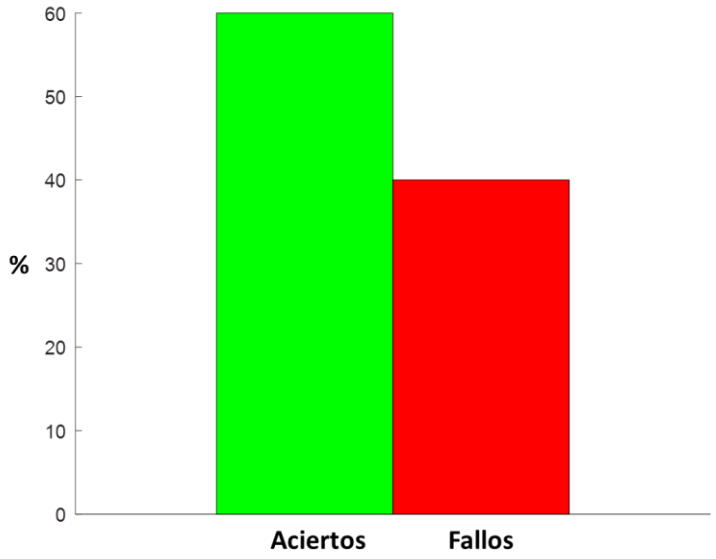

Figura 33: Porcentaje de aciertos y fallos sobre tierra roja

A la vista de los resultados obtenidos, en esta versión se tiende a reconocer correctamente las predicciones que se realizan con imágenes con un fondo rojo, aunque también se reconocen algunas con otro tipo de fondo como en el caso de las figuras con fondo verde. Hay que tener en cuenta que estos resultados se han obtenido utilizando un modelo entrenado solamente con muestras con un fondo rojo y un total de veintiséis imágenes.

5.3 VERSIÓN SOBRE CÉSPED

Las imágenes utilizadas para el entrenamiento de esta versión son del tipo definido por la Figura 24 y la Figura 29. Tal y como se indica en la Tabla 1, se han utilizado 44 imágenes para la “Plataforma 1” y otras 44 para la “Plataforma 2”.

Los parámetros utilizados para el entrenamiento son los siguientes:

- Épocas: 10
- Ratio de aprendizaje: 10^{-4}
- Unidades de densidad: 100
- Fracción de tamaño de lote: 4/10

Los resultados de la clasificación obtenidos para cada una de las imágenes mostradas en la Figura 23 hasta la Figura 32 se muestran en la Tabla 3 y la Figura 34.

Tabla 3: Resultados de la clasificación para versión sobre césped

FIGURA	RESULTADO
Figura 23: Plataforma 1 Tierra Roja	Correcto
Figura 24: Plataforma 1 Césped	Correcto
Figura 25: Plataforma 1 Madera	Correcto
Figura 26: Plataforma 1 Gravilla	Incorrecto
Figura 27: Plataforma 1 Tierra	Incorrecto
Figura 28: Plataforma 2 Tierra Roja	Correcto
Figura 29: Plataforma 2 Césped	Correcto
Figura 30: Plataforma 2 Gravilla	Incorrecto
Figura 31: Plataforma 2 Tierra	Correcto
Figura 32: Plataforma 2 Madera	Correcto

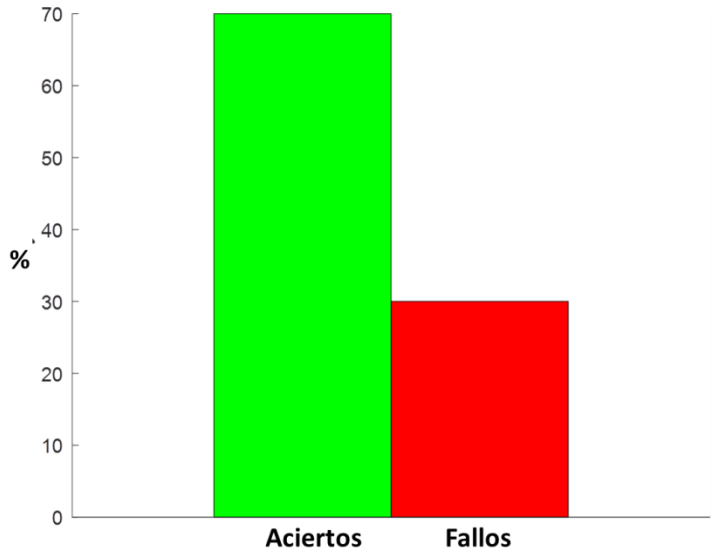

Figura 34: Porcentaje de aciertos y fallos sobre césped

Al igual que en la versión anterior, hay que tener en cuenta que solamente se han utilizado imágenes con un fondo verde para el entrenamiento. Se puede observar en este caso que se produce un error menos que en el caso anterior. Este resultado tiene sentido cuando se analizan los resultados de la versión sobre tierra roja, donde también se comprueba que no hubo errores al realizar clasificaciones con imágenes con fondo verde y en esta versión no hubo errores al realizar predicciones con imágenes con fondo rojo.

5.4 VERSIÓN SOBRE GRAVILLA

De nuevo, las imágenes utilizadas para el entrenamiento de esta versión son del tipo definido por la Figura 26 y la Figura 30. Tal y como se indica en la Tabla 1, se han utilizado 40 imágenes para la “Plataforma 1” y otras 40 para la “Plataforma 2”.

Los parámetros utilizados para el entrenamiento son los siguientes:

- Épocas: 10
- Ratio de aprendizaje: 10^{-4}
- Unidades de densidad: 100
- Fracción de tamaño de lote: 4/10

Los resultados de la clasificación obtenidos para cada una de las imágenes mostradas en la Figura 23 hasta la Figura 32 se muestran en la Tabla 4 y la Figura 35.

Tabla 4: Resultados de la clasificación para versión sobre gravilla

FIGURA	RESULTADO
Figura 23: Plataforma 1 Tierra Roja	Incorrecto
Figura 24: Plataforma 1 Césped	Incorrecto
Figura 25: Plataforma 1 Madera	Correcto
Figura 26: Plataforma 1 Gravilla	Correcto
Figura 27: Plataforma 1 Tierra	Correcto
Figura 28: Plataforma 2 Tierra Roja	Incorrecto
Figura 29: Plataforma 2 Césped	Correcto
Figura 30: Plataforma 2 Gravilla	Correcto
Figura 31: Plataforma 2 Tierra	Incorrecto
Figura 32: Plataforma 2 Madera	Incorrecto

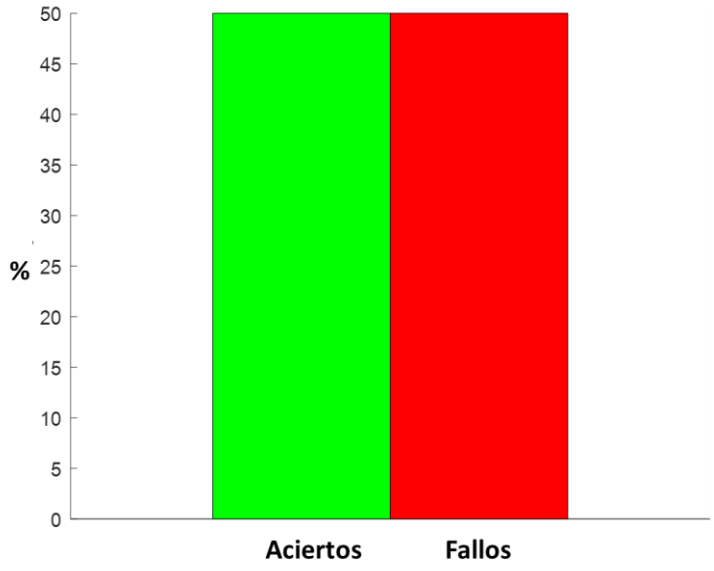

Figura 35: Porcentaje de aciertos y fallos sobre gravilla

Como se puede observar, es en esta versión donde más fallos se han producido. Esto se debe a varios factores entre los que tienen especial importancia la altura con la que se ha captado la imagen y los ángulos. Al observar detenidamente las muestras utilizadas se puede observar que estas imágenes son las que más profundidad tienen, además de ser imágenes tomadas desde ángulos muy diferentes. Para resolver este problema sería necesario disponer de un número mayor de muestras de este tipo.

5.5 VERSIÓN COMPLETA

En las versiones anteriores se realizan entrenamientos con imágenes específicas para cada caso, en cambio en esta versión se han realizado varios entrenamientos utilizando todas las muestras disponibles en cada versión anterior y un extra de muestras sobre tierra y madera.

Con el fin de cumplir los objetivos de este proyecto, el dron tiene que ser capaz de reconocer la plataforma de aterrizaje independientemente del terreno en el que se encuentre, por ello necesitamos un modelo capaz de detectar cualquier versión anterior y otras opciones que no se contemplan en las versiones anteriores. Para llevar a cabo esto se han realizado varios entrenamientos sobre el mismo modelo, ya que, al realizar un entrenamiento el progreso se guarda por tanto cuantos más entrenamientos se realicen, más "aprenderá" el modelo.

Primer entrenamiento: Se han utilizado los parámetros y las muestras de la versión sobre tierra roja:

- Épocas: 10
- Ratio de aprendizaje: 1/10000
- Unidades de densidad: 100
- Fracción de tamaño de lote: 4/10

Tabla 5: Resultados de la clasificación para el primer entrenamiento completo

FIGURA	RESULTADO
Figura 23: Plataforma 1 Tierra Roja	Correcto
Figura 24: Plataforma 1 Césped	Correcto
Figura 25: Plataforma 1 Madera	Correcto
Figura 26: Plataforma 1 Gravilla	Incorrecto
Figura 27: Plataforma 1 Tierra	Correcto
Figura 28: Plataforma 2 Tierra Roja	Correcto
Figura 29: Plataforma 2 Césped	Incorrecto
Figura 30: Plataforma 2 Gravilla	Correcto
Figura 31: Plataforma 2 Tierra	Correcto
Figura 32: Plataforma 2 Madera	Incorrecto

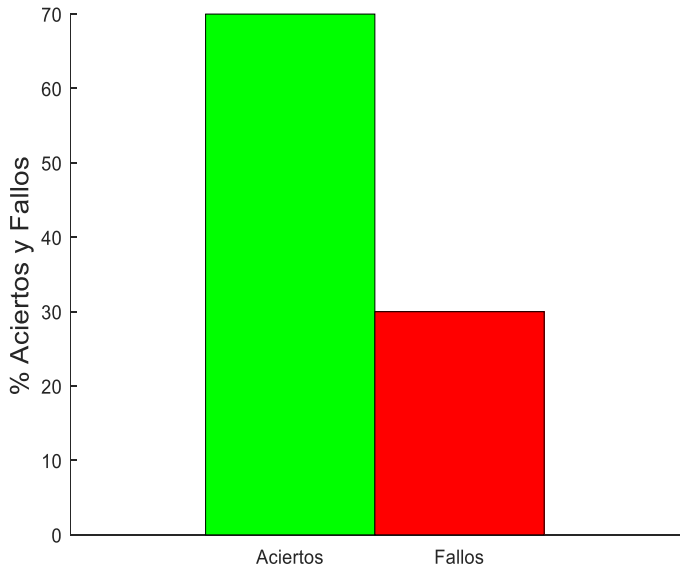

Figura 36: Porcentaje de aciertos y fallos primer entrenamiento

A diferencia de la versión sobre tierra roja, hay un fallo menos en este caso. Este fenómeno se debe a que las muestras no se utilizan siempre en el mismo orden, aunque el usuario las introduzca en orden, antes de realizar el entrenamiento, las muestras se reordenan de forma aleatoria cada vez.

Segundo entrenamiento: Se han utilizado los parámetros y las muestras de la versión sobre césped:

- Épocas: 10
- Ratio de aprendizaje: 1/9913
- Unidades de densidad: 100
- Fracción de tamaño de lote: 4/10

Tabla 6: Resultados de la clasificación para el segundo entrenamiento completo

FIGURA	RESULTADO
Figura 23: Plataforma 1 Tierra Roja	Correcto
Figura 24: Plataforma 1 Césped	Correcto
Figura 25: Plataforma 1 Madera	Incorrecto
Figura 26: Plataforma 1 Gravilla	Incorrecto
Figura 27: Plataforma 1 Tierra	Correcto
Figura 28: Plataforma 2 Tierra Roja	Correcto
Figura 29: Plataforma 2 Césped	Correcto
Figura 30: Plataforma 2 Gravilla	Correcto
Figura 31: Plataforma 2 Tierra	Correcto
Figura 32: Plataforma 2 Madera	Correcto

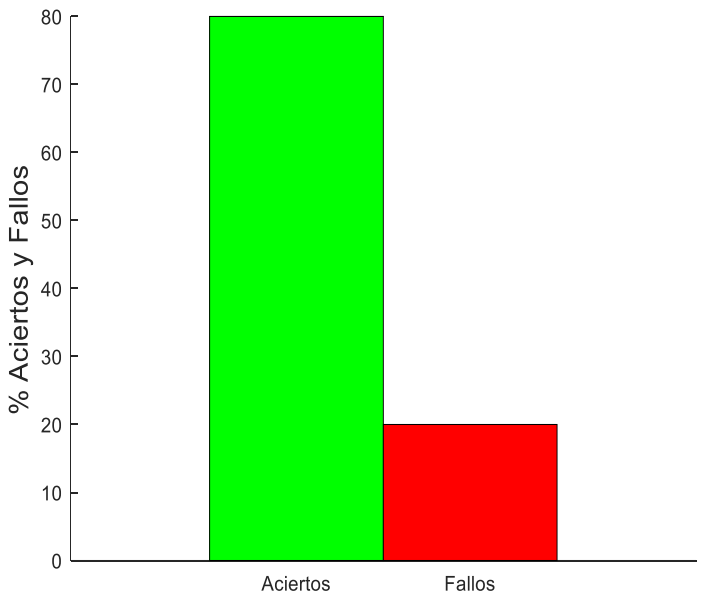

FIgura 37: Porcentaje de aciertos y fallos segundo entrenamiento

Al proporcionar más muestras la red neuronal cambia, por tanto, los resultados también. En este caso el entrenamiento ha dado lugar a una mejora a la hora de realizar las predicciones.

Tercer entrenamiento: Se han utilizado los parámetros y las muestras de la versión sobre gravilla:

- Épocas: 10
- Ratio de aprendizaje: 1/10000
- Unidades de densidad: 100
- Fracción de tamaño de lote: 4/10

Tabla 7: Resultados de la clasificación para el tercer entrenamiento completo

FIGURA	RESULTADO
Figura 23: Plataforma 1 Tierra Roja	Correcto
Figura 24: Plataforma 1 Césped	Correcto
Figura 25: Plataforma 1 Madera	Correcto
Figura 26: Plataforma 1 Gravilla	Correcto
Figura 27: Plataforma 1 Tierra	Incorrecto
Figura 28: Plataforma 2 Tierra Roja	Incorrecto
Figura 29: Plataforma 2 Césped	Correcto
Figura 30: Plataforma 2 Gravilla	Correcto
Figura 31: Plataforma 2 Tierra	Incorrecto
Figura 32: Plataforma 2 Madera	Incorrecto

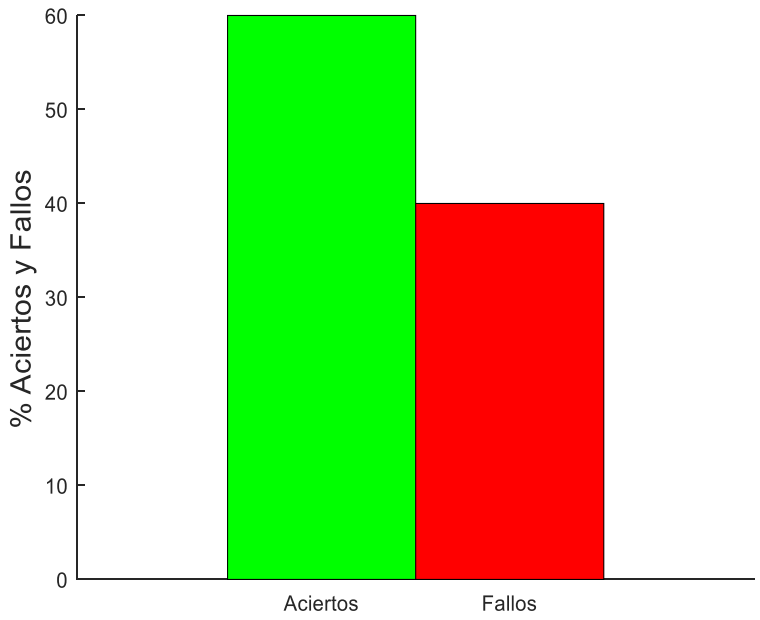

FIgura 38: Porcentaje de aciertos y fallos tercer entrenamiento

Hay que tener en cuenta que al entrenar la versión sobre gravilla se llegó a la conclusión de que era la versión con más muestras dispares, con diferentes ángulos y alturas por tanto es normal que al añadir dichas muestras la red “aprenda” menos.

Cuarto entrenamiento: Se han utilizado diferentes tipos de muestras, un total de trescientas dos imágenes, ciento cincuenta y uno correspondiente a cada plataforma:

- Épocas: 10
- Ratio de aprendizaje: 1/9913
- Unidades de densidad: 100
- Fracción de tamaño de lote: 4/10

Tabla 8: Resultados de la clasificación para el cuarto entrenamiento completo

FIGURA	RESULTADO
Figura 23: Plataforma 1 Tierra Roja	Incorrecto
Figura 24: Plataforma 1 Césped	Correcto
Figura 25: Plataforma 1 Madera	Correcto
Figura 26: Plataforma 1 Gravilla	Correcto
Figura 27: Plataforma 1 Tierra	Correcto
Figura 28: Plataforma 2 Tierra Roja	Incorrecto
Figura 29: Plataforma 2 Césped	Correcto
Figura 30: Plataforma 2 Gravilla	Correcto
Figura 31: Plataforma 2 Tierra	Correcto
Figura 32: Plataforma 2 Madera	Correcto

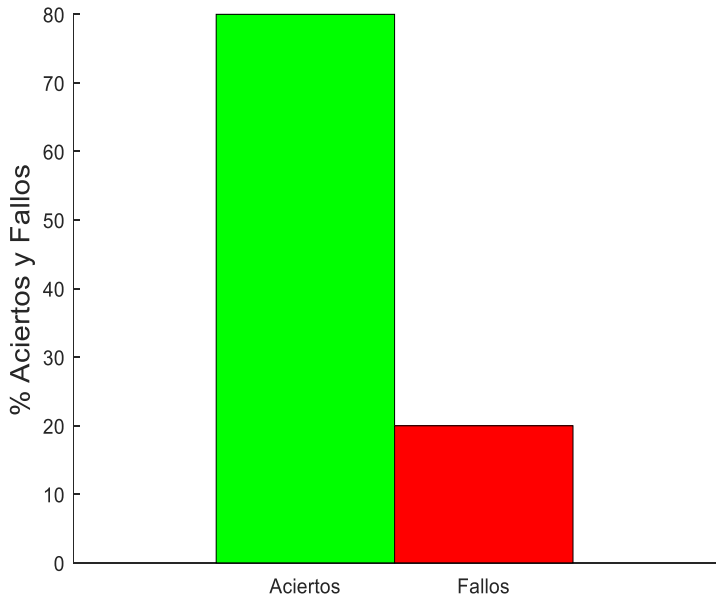

FIgura 39: Porcentaje de aciertos y fallos cuarto entrenamiento

Como se puede observar, los resultados proporcionados por la red neuronal han vuelto a cambiar, esta vez a mejor. En total se han utilizado doscientas cuarenta y cuatro imágenes para cada tipo de plataforma. Para obtener más fiabilidad es necesario disponer de más muestras principalmente, aunque hay que tener en cuenta otros factores como por ejemplo los parámetros especificados para los entrenamientos, no sobreentrenar el modelo y la calidad de las muestras.

Gracias a esta versión completa se ha podido comprobar que, en función de las muestras proporcionadas, “aprende” aunque a veces se puede confundir gracias a entrenamientos con muestras de poca calidad o introduciendo un mayor número de muestras de un tipo que de otro, como ha sido el caso de las muestras con tierra roja en el último entrenamiento.

6 CONCLUSIONES Y TRABAJO FUTURO

Tras el desarrollo del proyecto y por tanto de la aplicación, junto con las pruebas realizadas, a continuación, se describen las conclusiones obtenidas, a la vez que se proporcionan ideas sobre la evolución en el futuro de cara a conseguir mejoras significativas.

6.1 CONCLUSIONES

El objetivo principal del proyecto era la creación de una herramienta básica capaz de gestionar diferentes modelos en el ámbito de RNC con posibilidad de creación de los mismos y con vistas a su entrenamiento y posterior reconocimiento de objetos en general, focalizando el desarrollo en el reconocimiento de plataformas de aterrizaje para VANT.

Se puede afirmar que se ha cumplido este objetivo, de forma que la aplicación permite crear diferentes modelos dentro de una categoría, eliminarlos y entrenarlos progresivamente, para realizar posteriores clasificaciones.

Por otro lado, se ha podido comprobar el rendimiento de un determinado modelo realizando varios experimentos consistentes en entrenamientos con diferentes muestras; ya sea, tanto en tamaño como en calidad, aparte de comprobar la evolución de un modelo de RNC en función de los entrenamientos realizados.

Además del objetivo principal, se han cumplido otros objetivos específicos planteados en el ámbito del presente proyecto, como por ejemplo analizar las diferentes tecnologías ya sean librerías o plataformas que permiten utilizar algoritmos y metodologías para crear aplicaciones dentro del paradigma de aprendizaje profundo y más concretamente bajo el ámbito de las RNC. A esto hay que añadir el hecho de haber integrado los desarrollos bajo tecnologías Web, desplegando la aplicación en un navegador, evolucionando hacia un servidor web con su ejecución en distintos dispositivos, ya sean de escritorio o móviles, como se puede observar en la Figura 40.

Figura 40: Ejecución de la aplicación en dispositivo móvil

Desde el punto de vista operativo y en relación a las metodologías de trabajo, conviene señalar el hecho de que el proyecto se ha desarrollado en equipo de forma colaborativa, lo que ha permitido establecer la organización apropiada para desarrollar las tareas.

6.2 Trabajo Futuro

Como opciones de futuro surgidas a lo largo del desarrollo del proyecto y en las distintas fases de ejecución del mismo cabe señalar las siguientes acciones:

- Mejorar la interfaz de usuario para que sea más amigable y más sencilla. En este sentido, además sería beneficioso añadir indicadores de progreso para que el usuario sepa correctamente en qué estado se encuentra la ejecución del programa o durante la carga de imágenes.

- Mejorar el control y visualización de los errores que pueda lanzar la aplicación.

- Realizar una gestión de usuarios y contraseñas, de tal manera que se permita al inicio de sesión cargar los modelos propios de cada usuario. La gestión relativa a la compartición de modelos y versiones con diferentes permisos constituye otra posibilidad de mejora, tal es el caso de permisos para creación o eliminación de versiones y realizar el entrenamiento. Las clasificaciones deberían poder realizarse siempre que un usuario comparta a otro su modelo y versión.

- En la clasificación convendría añadir información relativa al grado de pertenencia de la imagen de entrada a una clase dada, no sólo la clase elegida. Para lo cual sería necesario derivar el resultado de la última capa de las RNC, generalmente la que proyecta los valores de las neuronas a valores restringidos al rango [0,1], denominada generalmente *softmax* (Goodfellow y col., 2016).

- Añadir la posibilidad de que el usuario introduzca de manera gráfica las capas que quiera tener en su modelo, de tal manera que por defecto pueda trabajar con el modelo directamente, tal y como se importa, pero ampliando la posibilidad para cambiar las capas del modelo durante la creación, de esta forma el usuario tendría más flexibilidad para poder realizar los propios entrenamientos con imágenes, a la vez que se puede conseguir mayor precisión en las clasificaciones.

- Añadir la opción de exportar el modelo. De esta forma, el usuario podría descargar su modelo en formato JSON y transportarlo a su propio sistema mientras posea integración con TensorflowJS (2019).

- Realizar el entrenamiento mediante el suministro de secuencias de video. Actualmente la aplicación permite un único tipo de entrenamiento básico mediante imágenes en formato .jpg y .png. Esto limita la capacidad de entrenamiento para un número elevado de muestras, a la vez que resulta tedioso. Para resolver este problema, se puede utilizar funcionalidad ya implementada en TensorflowJS (2019) y en el motor de JavaScript de los navegadores para conectar ambas librerías a una cámara y realizar el entrenamiento utilizando los *frames* del vídeo. No obstante, esto

sería, en principio independiente de la cámara instalada en el VANT por cuestiones de comunicación. Se utilizaría con las capturas de video realizadas por dicha cámara.

- Realizar clasificaciones múltiples. La aplicación permite realizar la clasificación de una única imagen cada vez. Al igual que en el caso de los entrenamientos, cabe la posibilidad de proporcionar un conjunto de imágenes o secuencias en formato vídeo para realizar las clasificaciones mediante una única acción de selección.

- Mejorar el rendimiento del servidor utilizando funcionalidades más complejas ofrecidas por TensorflowJS (2019), como por ejemplo utilizar la GPU en lugar de la CPU a la hora de realizar el procesamiento de las imágenes y los entrenamientos.

- Mejorar el rendimiento utilizando de forma más eficiente el liberador de memoria que ofrece la API de TensorflowJS (2019).

- Estudiar la posibilidad de desplegar la aplicación o parte de ella para su ejecución en aceleradores específicos del tipo Tensor Processing Units (TPUs) (2019).

- Desde el punto de vista del conjunto de imágenes, se plantea la creación de un conjunto de datos de imágenes añadiendo más ejemplos con capturas a distintas alturas, ángulos de inclinación, diferentes condiciones de iluminación y vistas parciales de la plataforma.

- Crear modelos con más clases, que no contengan la plataforma y que describan características propias de los entornos de exterior donde debe operar el VANT.

- Proporcionar información sobre el grado de ajuste de la función de pérdida durante la fase de minimización, para determinar el grado de precisión del ajuste y por consiguiente la validación del modelo en fase de entrenamiento.

- Ampliar la aplicación para dar cabida a la inclusión de técnicas basadas en RNC, pero con la orientación de segmentación de regiones en lo que se conoce exactamente como R-RNC, donde la primera R hace referencia concretamente al concepto de Región.

7 CONCLUSIONS AND FUTURE WORK

After the development of the project and therefore of the application, together with the tests carried out, the conclusions obtained are described below, while ideas about the evolution in the future are provided in order to achieve significant improvements.

7.1 CONCLUSIONS

The main objective of the project was the creation of a basic tool capable of managing different models in the field of the RNC with the possibility of creating them and with a view to their training and subsequent recognition of objects in general, focusing the development on the recognition of landing platforms for UAVs.

It can be affirmed that this objective has been fulfilled, so that the application allows to create different models within a category, eliminate them and train them progressively, to make further classifications.

On the other hand, it has been possible to verify the performance of a certain model by performing several experiments consisting of training with different samples; either in size or in quality, apart from checking the evolution of an RNC model according to the training carried out.

In addition to the main objective, other specific objectives have been met in the field of this project, such as analyzing different technologies whether as libraries or platforms which allow the use of algorithms and methodologies to create applications within the deep learning paradigm and more specifically under the field of the RNC. To this, it must be added the fact of having integrated the developments under Web technologies, deploying the application in a browser, evolving into a web server with its execution on different devices, whether as desktop or mobile phones.

Figura 41: Application execution on a mobile device

From the operational point of view and in relation to teamwork methodologies, it is worth noting the fact that the project has been developed collaboratively as a team, which made it possible to establish the appropriate organization to carry out the different tasks.

7.2 FUTURE WORK

As future options emerged during the development of the project and in the different phases of its execution, the following actions should be noted:

- Improve the user interface to make it more friendly and easy to use. In this sense, it would also be beneficial to add progress indicators so that the user knows correctly in what state the execution of the program is or, moreover, during the loading of images

- Improve the control and visualization of errors that the application may launch.

- Perform a management of users and passwords, in such way that at the beginning of the session it is possible to load the models of each user. The management regarding the sharing of models and versions with different permissions constitutes another possibility of improvement, such as the case of permissions for creation or elimination of versions and carrying out the training. Classifications should be possible whenever a user shares his model and version with other users.

- In the classification it would be convenient to add information regarding the degree of belonging of the input image to a given class, not just the chosen class. For which it would be necessary to lead the result of the last layer of the RNC, generally the one that projects the values of the neurons to values restricted to the range [0,1], usually called softmax (Goodfellow y col., 2016).

- Add the possibility that the user graphically enter the layers he want to have in his model, so that by default he can work with the model directly, as it is imported, but expanding the possibility to change the layers of the model during the creation; in this way the user will have more flexibility to be able to perform their own training with images, while achieving at the same time greater precision in the classifications.

- Add the option to export the model. In this way, the user could download their model in JSON format and transport it to their own system while having integration with TensorflowJS (2019).

- Carry out the training throughout the supply of video sequences. Currently, the application allows a single type of basic training through images in .jpg and .png format. This limits the training capacity for a large number of samples, and at the same time is tedious. To solve this problem, you can use functionality already implemented in TensorflowJS (2019) and in the JavaScript engine of the browsers to connect both libraries to a camera and perform the training using the "frames" of the video. However, this would be, at first, independent from the camera installed in the UAV for communication reasons. It would be used with the video captures made by the mentioned camera.

- Make multiple classifications. The application allows the classification of a single image each time. As in the case of training, it is possible to provide a set of images or sequences in a video format to make the classifications throughout a single selection action.

- Improve server efficiency by using more complex functionalities offered by TensorflowJS (2019), such as, for example, using the GPU instead of the CPU when processing images and training.

- Improve efficiency by using, in a more effective way the memory release functions offered by the TensorflowJS (2019) API.

- Study the possibility of deploying the application or part of it for its execution in specific accelerators of the Tensor Processing Units (TPUs) (2019) type.

- From the point of view of the set of images, the creation of a set of image data is proposed, adding more examples with captures at different heights, angles, different lighting conditions and partial views of the platform.

- Create models with more classes, which do not contain the platform and that describe the different characteristics of the external environments where the UAV should operate.

- Provide information of the degree of adjustment of the loss function during the minimization phase, to determine the degree of accuracy of the adjustment and therefore the validation of the model in the training phase.

- Expand the application to allow the inclusion of techniques based on RNC but with the orientation of segmentation of regions in what is known exactly as R-RNC, where the first R refers specifically to the concept of Region.

8 BIBLIOGRAFÍA

1. Accord.Net (2018). Disponible en su web oficial: http://accord-framework.net/

2. Activation Functions in Neural Networks (2019): https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

3. Angular 6 (2019). Disponible en su web oficial: https://angular.io/

4. AlexNet (2018). ImageNet Classification with Deep Convolutional Neural Networks: https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/

5. AprilTag. Disponible on-line: https://april.eecs.umich.edu/software/apriltag.html. (accedido marzo 2019).

6. Araar, O., Aouf N., Vitanov I. (2017). Vision based autonomous landing of multirotor uav on moving platform. Journal of Intelligent & Robotic System 2017, 85, 369–384.

7. Arroyo, L., Caballero, D., Puerro, E. (2017). Reconocimiento de una plataforma para aterrizaje de UAVs mediante procesamiento de imágenes. Trabajo Fin de Grado. Universidad Complutense de Madrid. Disponible on-line: https://eprints.ucm.es/44699/ (accedido marzo 2019).

8. ArUco (2019). Detection of ArUco Markers. Disponible on-line: http://docs.opencv.org/trunk/d5/dae/tutorial_aruco_detection.html (accedido marzo 2019).

9. Awesome-Typescript-Loader (2019), disponible en su GitHub oficial: https://github.com/s-panferov/awesome-typescript-loader

10. Bay, H., Ess, A., Tuytelaars, T., van Gool, L. (2008). SURF: Speeded Up Robust Features. Computer Vision and Image Understanding (CVIU), vol. 110, No. 3, pp. 346-359.

11. Bootstrap (2019). Disponible en su web oficial: https://getbootstrap.com/

12. Boureau, Y., Bach, F., LeCun, Y., and Ponce, J. (2010a). Learning mid-level features for recognition. In Proc. IEEE Int. Conference on Computer Vision and Pattern Recognition (CVPR'10).

13. Boureau, Y., Ponce, J., and LeCun, Y. (2010b). A theoretical analysis of feature pooling in vision algorithms. In Proc. IEEE Int. Conference on Machine learning (ICML'10).

14. Chaves, S.M., Wolcott R.W., Eustice R.M. (2015). NEEC research: Toward GPS–denied landing of unmanned aerial vehicles on ships at sea. Nav. Eng. J., 127, 23–35.

15. Chen J., Miao X., Jiang H., Chen J., Liu X. (2017). Identification of Autonomous Landing Sign for Unmanned Aerial Vehicle Based on Faster Regions with Convolutional Neural Network. IEEE International Conference on Chinese Automation Congress (CAC), pp. 2019-2114.

16. Caltech 101 (2018). Disponible en su web oficial: http://www.vision.caltech.edu/Image_Datasets/Caltech101/

17. Caltech 256 (2018). Disponible en su web oficial: http://www.vision.caltech.edu/Image_Datasets/Caltech256/

18. Canvas (2019). Disponible en su web oficial: https://canvasjs.com/

19. CentOs (2019). Disponible en su web oficial: https://www.centos.org/

20. Cloud Tensor Processing Units (TPUs) (abril de 2019). Disponible en su web oficial: https://cloud.google.com/tpu/docs/tpus

21. Cocchioni F., Mancini A., Longhi S. (2014). Autonomous navigation, landing and recharge of a quadrotor using artificial vision. International conference on unmanned aircraft systems (ICUAS), pp. 418–429.

22. Commit Conf (23 y 24 de noviembre de 2018): https://2018.commit-conf.com/

23. Conferencia sobre TensorflowJS de la “Commit Conf” en video disponible en Youtube: https://www.youtube.com/watch?time_continue=979&v=RN_DLgao4TU

24. Cruz, J.M., Sánchez, B., Pajares, G. (2012). System for guiding an unmanned vehicle towards a platform using visual analysis. Patent nº 201001592, 2013.

25. Difference between softmax function and sigmoid function: http://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/

26. Dumoulin, V., Visin, F. (2016) A guide to convolution arithmetic for deep learning. arXiv:1603.07285v2. Disponible on-line: https://arxiv.org/abs/1603.07285 (accedido Febrero 2019).

27. Electron (2019). Disponible en su web oficial: https://electronjs.org/

28. Express (2019). Disponible en su web oficial: https://expressjs.com/es/

29. FileSystem de Node (2019). Disponible en la web oficial de NodeJS: https://nodejs.org/api/fs.html

30. García-Pulido, J.A., Pajares, G., Dormido, S., de la Cruz, J.M. (2017). Recognition of a landing platform for unmanned aerial vehicles by using computer vision-based techniques. Expert Systems with Applications, 76,152-165.

31. Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F.J., Marín-Jiménez, M.J. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition, 47(6), 2280–2292.

32. Gentle introduction to the Adam Optimization Algorithm for Deep Learning: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

33. Git (2018). Disponible en la web oficial: https://git-scm.com/

34. Github (2018). Disponible en la web oficial: https://github.com/

35. Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep learning. MIT Press. Disponible on-line: https://www.deeplearningbook.org/ (accedido Febrero 2019).

36. Guili X., Yong Z., Shengyu J., Yuehua CH., Yupeng T. (2009). Research on computer vision-based for UAV autonomous landing on a ship. Pattern Recognition Letters, 30(6), 600–605.

37. ImageNet (2018). Disponible en su web oficial: http://www.image-net.org/

38. Implementing YOLO v3 Tensorflow (TF-SLIM) (2018). Disponible on-line en el siguiente enlace: https://itnext.io/implementing-yolo-v3-in-tensorflow-tf-slim-c3c55ff59dbe

39. Kyristsis, S., Antonopoulos, A., Chanialakis, T., Stefanakis, E., Linardos, C., Tripolitsiotis A., Partsinevelos P. (2016). Towards autonomous modular UAV missions: The detection, geolocation and landing paradigm. Sensors, 16(11), 1844.

40. Lange, S., Sünderhauf, N., Protzel, P. (2008). Autonomous landing for a multirotor uav using vision. Workshop Proc. of SIMPAR 2008 Intl. Conf. on Simulation, Modelling and Programming for Autonomous Robots, pp. 482–491.

41. Lange, S., Sunderhauf, N., Protzel, P. (2009). A vision based onboard approach for landing and position control of an autonomous multirotor uav in gps-denied environments. Advanced Robotics, 2009. ICAR 2009. International Conference on. IEEE, 2009, pp. 1–6.

42. LeCun, Y. (1989). Generalization and network design strategies. Technical ReportCRG-TR-89-4, University of Toronto. 326, 345747. Disponible on-line: http://yann.lecun.com/exdb/publis/pdf/lecun-89.pdf (accedido febrero 2019).

43. Ling, K. (2014). Precision landing of a quadrotor uav on a moving target using low–cost sensors. Master Thesis, University of Waterloo, Canada. UWSpace. Disponible on-line: https://uwspace.uwaterloo.ca/handle/10012/8803. (accedido marzo 2019).

44. Li, Y., Wang, Y., Luo, H., Chen Y., Jiang Y. (2012). Landmark recognition for UAV autonomous landing based on vision. Application Research of Computers 7, pp. 2780–2783.

45. Mercedes Benz, sistema PRE-SAFE (mayo de 2019). Proporciona la información de su sistema en su web oficial: https://www.mercedes-benz.es/passengercars/mercedes-benz-cars/models/s-class/mercedes-maybach/explore/intelligent-drive-technologies/pre-safe-system.popup.html

46. Microsoft ASP .Net (2018). Disponible en la web oficial: https://dotnet.microsoft.com/apps/aspnet

47. Microsoft ASP .Net Web APIs (2018). Disponible en la web oficial: https://dotnet.microsoft.com/apps/aspnet/apis

48. Microsoft Azure (2018). Disponible en la web oficial: https://azure.microsoft.com/es-es/

49. Microsoft Azure App Service (2018). Disponible en la web oficial: https://azure.microsoft.com/es-es/services/app-service/

50. Microsoft Azure Blob Storage (2018). Disponible en la web oficial: https://azure.microsoft.com/es-es/services/storage/blobs/

51. Microsoft Azure Virtual Machines (2019). Disponible en la web oficial: https://azure.microsoft.com/es-es/services/virtual-machines/

52. Microsoft CNTK (2018). Disponible en su web oficial https://docs.microsoft.com/en-us/cognitive-toolkit/

53. Microsoft Cognitive Services (2018). Disponible en su web oficial: https://azure.microsoft.com/es-es/services/cognitive-services/

54. Microsoft Custom Vision (2018). Disponible en su web oficial: https://www.customvision.ai/

55. Microsoft SQL Server (2018). Disponible en la web oficial: https://www.microsoft.com/es-es/sql-server

56. Microsoft Universal Windows Platforms (2018). Disponible en su web oficial: https://docs.microsoft.com/es-es/windows/uwp/get-started/universal-application-platform-guide

57. Microsoft Visual Studio 2017 (2018). Disponible en la web oficial: https://visualstudio.microsoft.com/es/

58. Microsoft Visual Studio Code (2019). Disponible en la web oficial: https://code.visualstudio.com/

59. Microsoft Windows Forms (2018). Disponible en la web oficial: https://docs.microsoft.com/es-es/dotnet/framework/winforms/

60. Microsoft .Net Framework (2018). Disponible en su web oficial: https://docs.microsoft.com/es-es/dotnet/framework/

61. Multer (2019). Disponible en GitHub: https://github.com/expressjs/multer

62. Netbeans (2019). IDE para desarrollo. Disponible en su web: https://netbeans.org/kb/73/ide/javascript-editor.html

63. Nguyen, P.H., Arsalan M., Koo, J.H., Naqvi, R.A., Truong, N.Q., Park, K.R. (2018). LightDenseYOLO: A Fast and Accurate Marker Tracker for Autonomous UAV Landing by Visible Light Camera Sensor on Drone. Sensors, 18(6), 1703.

64. NodeJS (2019). Disponible en su web oficial: https://nodejs.org/es/

65. Nodemon (2019). Disponible en su web oficial: https://nodemon.io/

66. npm (2019). Disponible en su web oficial: https://www.npmjs.com/

67. Olson, E. (2011). AprilTag: A robust and flexible visual fiducial system. IEEE International Conference on Robotics and Automation (ICRA), 2011.

68. Pajares, G. (2015). Overview and Current Status of Remote Sensing Applications Based on Unmanned Aerial Vehicles (UAVs). Photogrammetric Engineering and Remote Sensing, 81(4), 281-330.

69. Pexels (2018). Disponible en su web oficial: https://www.pexels.com/

70. Piñeiro, E. (2016) Reconocimiento de una plataforma para aterrizaje de UAV mediante procesamiento de imágenes. Trabajo Fin de Grado. Universidad Complutense de Madrid. Disponible on-line: https://eprints.ucm.es/39845/ (accedido marzo 2019).

71. Pixel (2019). Disponible en la web de npm: https://www.npmjs.com/package/pixel

72. Pm2 (2019). Disponible en su web oficial: http://pm2.keymetrics.io/

73. Polvara, R., Patacchiola, M., Sharma, S., Wan, J., Manning, A., Sutton, R., Cangelosi, A. (2017). Autonomous quadrotor landing using deep reinforcement learning. arXiv preprint arXiv:1709.03339.

74. React (2019). Disponible en su web oficial: https://reactjs.org/

75. Saripalli, S., Montgomery, J.F., Sukhatme, G.S. (2012). Vision-based autonomous landing of an unmanned aerial vehicle. IEEE Int. Conf. on Robotics and Automation, 11-15, pp. 2799–2804.

76. Saripalli, S., Sukhatme, G. (2006). Landing on a moving target using an autonomous helicopter. Field and service robotics. Springer Tracts in Advanced Robotics, 2006, pp. 277–286.

77. Saxe, A., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. (2011). On random weights and unsupervised feature learning. In L. Getoor and T. Scheffer, editors, Proceedings of the 28th International Conference on Machine Learning (ICML-11), ICML ’11, pages 1089–1096, New York, NY, USA. ACM.

78. Sharp, C.S., Shakernia, O., Sastry, S.S. (2001). A vision system for landing an unmanned aerial vehicle. IEEE Int. Conf. on Robotics and Automation, pp. 1720–1727.

79. Sharp (2019). Disponible en su web oficial: https://sharp.dimens.io/en/stable/

80. Super Hexagon (2018). Disponible en Github: https://github.com/Schlipak/Hexagon.js/tree/master

81. Tensorflow (2019). Disponible en su web oficial: https://www.tensorflow.org

82. TensorflowJS (2019). Disponible en su web oficial: https://www.tensorflow.org/js

83. TensorflowJS API (2019): Disponible en su web oficial, la guía del framework utilizado: https://www.tensorflow.org/js/guide

84. TensorflowJS with MobileNet Github repository (2019). Disponible en Github: https://github.com/tensorflow/tfjs-examples/tree/master/mobilenet

85. Tesla Autopilot (mayo de 2019) sistema de conducción autónoma de la marca Tesla para sus vehículos, disponible en su web oficial: https://www.tesla.com/es_ES/autopilot

86. Typescript (2019). Disponible en su web oficial: https://www.typescriptlang.org/

87. Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names: https://gombru.github.io/2018/05/23/cross_entropy_loss/

88. Vue (2019). Disponible en su web oficial: https://vuejs.org/

89. Wang, L., Yang, Q., Guo, X. (2016). Recognition Algorithm of the Apron for Unmanned Aerial Vehicle Based on Image Corner Points. Laser Journal, 8, 71–74.

90. Webpack (2019). Disponible en su web oficial: https://webpack.js.org/

91. YOLO Real-Time Object Detection (2018): Disponible on-line en su web oficial: https://pjreddie.com/darknet/yolo/

92. Zhao, Y.J., Pei H.L. (2013). An improved vision–based algorithm for unmanned aerial vehicles autonomous landing. Applied Mechanics and Materials, 273, 560–565.

93. Zhou, Y. and Chellappa, R. (1988). Computation of optical flow using a neural network. Proc. IEEE International Conference on Neural Networks, 1988, pages 71–78.

9 ANEXO I: MANUAL DE DESPLIEGUE

A continuación, se procede a comentar todos y cada uno de los pasos necesarios para poder desplegar al completo la aplicación, en este punto se expondrá tanto la descripción de los pasos para desplegar la aplicación "B*ack-end"* como la de *"Front-end"*.

Para comenzar, se comenta en primer lugar, el despliegue del "*Back-end"* debido a que para que el front-end tenga toda su funcionalidad es necesario que esté desplegado y funcionando.

9.1 DESPLIEGUE DEL BACK-END

Para realizar el despliegue de la aplicación "*Back-end"*, son necesarios los siguientes elementos:

- Sistema operativo Windows, Linux o Mac OS.
- NodeJS (2019) instalado en el equipo.
- Conexión a internet.
- Un servidor Microsoft SQL Server (2018).

Una vez, se tengan los requisitos anteriores, se procederán a realizar los siguientes pasos:

1. Abrir el terminal de comandos del sistema operativo utilizado.
2. Acceder mediante el terminal de comandos al directorio donde se encuentre alojado el proyecto de "B*ack-end"*.
3. Ejecutar la instrucción "*npm install*", la cual descarga todas y cada una de las dependencias (indicadas en los archivo "*package.json*" y "*package-lock.json*") del proyecto para poder realizar el despliegue.
4. Crear un archivo denominado ".env" (ver Figura 42) en la raíz del proyecto, el cual se utilizará para establecer los datos de conexión con la base de datos y el puerto de la aplicación a ejecutar.

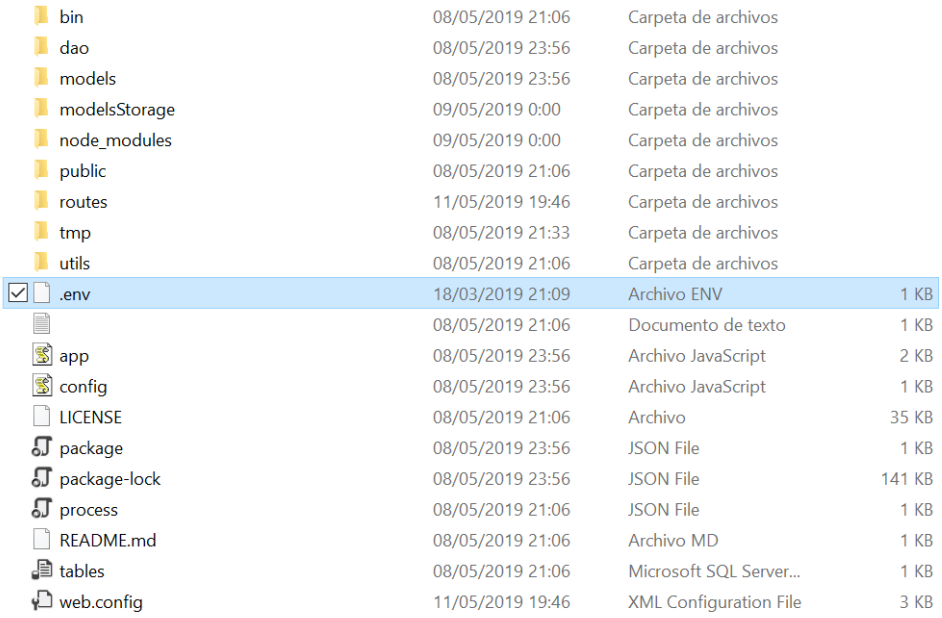

Figura 42: Archivo '.env' en la raíz del proyecto

5. Editar el archivo ".env" e incluir los siguientes datos, como se muestra en la Figura 43:

    - *databaseServer*: URL del servidor de base de datos.
    - *databaseName*: El nombre de la base de datos a la que conectar.
    - *databasePort*: El puerto de la base de datos.
    - *user*: Usuario de la base de datos.
    - *password*: Contraseña del usuario de base de datos.
    - *applicationPort*: El puerto donde se quiere ejecutar la aplicación web.

```
databaseServer = 'deeplearning.database.windows.net'
databaseName = 'DeepLearningDatabase'
databasePort = 1433
user = 'deeplearning'
password = 'TFGjaimeandrei!'
applicationPort = 2500
```

Figura 43: Contenido del archivo .env

6. Ejecutar la instrucción "*npm run start*", la cual, ejecutará la aplicación en el puerto indicado. Si la consola muestra los mensajes de la Figura 44 todo habrá ido bien y el servicio estará funcionando y esperando peticiones web.

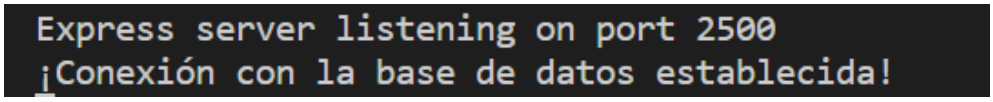

Figura 44: Salida correcta de consola de la ejecución de la aplicación

9.2 DESPLIEGUE DEL FRONT-END

A continuación, se describe el proceso necesario para realizar la compilación y ejecución de la aplicación frontal, para ello serán necesarios los siguientes elementos:

- Sistema operativo Windows, Linux o Mac OS.
- NodeJS (2019) instalado en el equipo.
- Conexión a internet.

Para realizar la compilación y la ejecución del código frontal, hay que seguir los siguientes pasos:

1. Abrir el terminal de comandos del sistema operativo utilizado.
2. Acceder mediante el terminal de comandos al directorio donde se encuentre alojado el proyecto de front-end.
3. Ejecutar la instrucción "*npm install*", la cual descargará todas y cada una de las dependencias del proyecto (indicadas en los archivo "*package.json"* y *"package-lock.json"*) para poder realizar la compilación y el despliegue.
4. Modificar el archivo: "/src/common/constants.tsx" para indicar la URL y el puerto del servidor "*Back-end"* de la siguiente manera:

   SERVICE_URL = 'http://{IP del servidor}:{Puerto de la aplicación}/model'

5. Una vez terminada la instalación de las dependencias y comprobada la URL del servidor, existen varias maneras de compilar y ejecutar la aplicación:

   - **Compilación y ejecución con** Electron (2019) **a la vez**: ejecutar la instrucción *"npm run start".*
   - **Compilación**: ejecutar la instrucción *"npm run compile".*
   - **Ejecución con electrón**: una vez ha sido compilado, se debe ejecutar la instrucción *"npm run electron"*.
   - **Ejecución en el navegador**: una vez ha sido compilado, se abre el archivo ./dist/pro/index.html con el navegador web.

10 ANEXO II: MANUAL DE USUARIO

Con el fin de ofrecer una experiencia de usuario más agradable se ha creado este manual. A continuación, se enumeran las funcionalidades disponibles y su utilización.

10.1 VISTA SELECCIONAR CATEGORÍA

Al iniciar la aplicación, por defecto aparecen todas las categorías almacenadas en la base de datos. Tal como se puede observar en la Figura 45.

Para que la aplicación gestione de forma correcta los modelos, es necesario que esté centralizada en un servidor; ya que, al hacer la ejecución en local la aplicación se conecta a la base de datos y podrá crear modelos; pero el almacenamiento de los ficheros de los modelos es local; por lo que si un usuario ejecuta la aplicación en su equipo, otro usuario la ejecuta en otro y ambos crean modelos, podrán ver los modelos, pero no podrán utilizarlos ya que el archivo del modelo lo almacena cada usuario en su propio equipo. Este problema desaparece si se utiliza un servidor centralizado y los usuarios acceden a esa aplicación.

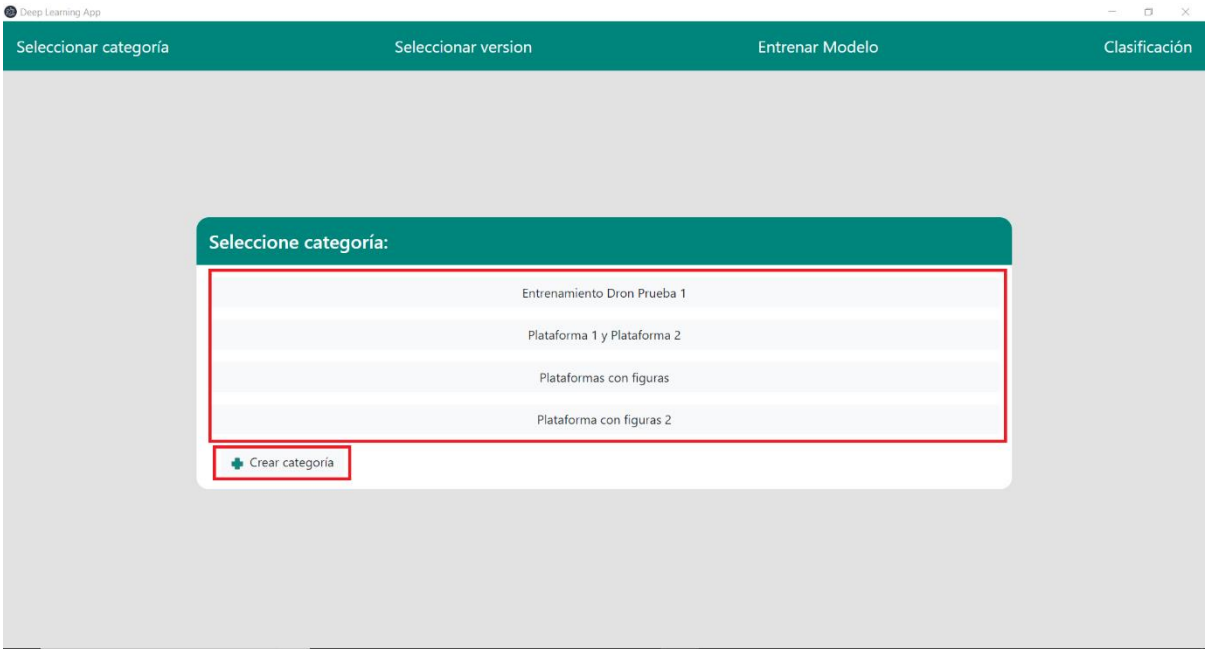

Figura 45: Vista de selección de categoría

Además de cargar todas las categorías existe la opción de crear una nueva categoría.

10.1.1 Crear categoría

Al hacer clic en “Crear categoría” automáticamente aparece el formulario de la Figura 46, el cual solicita un nombre para dicha categoría y también una confirmación de la creación mediante el botón “Crear categoría”.

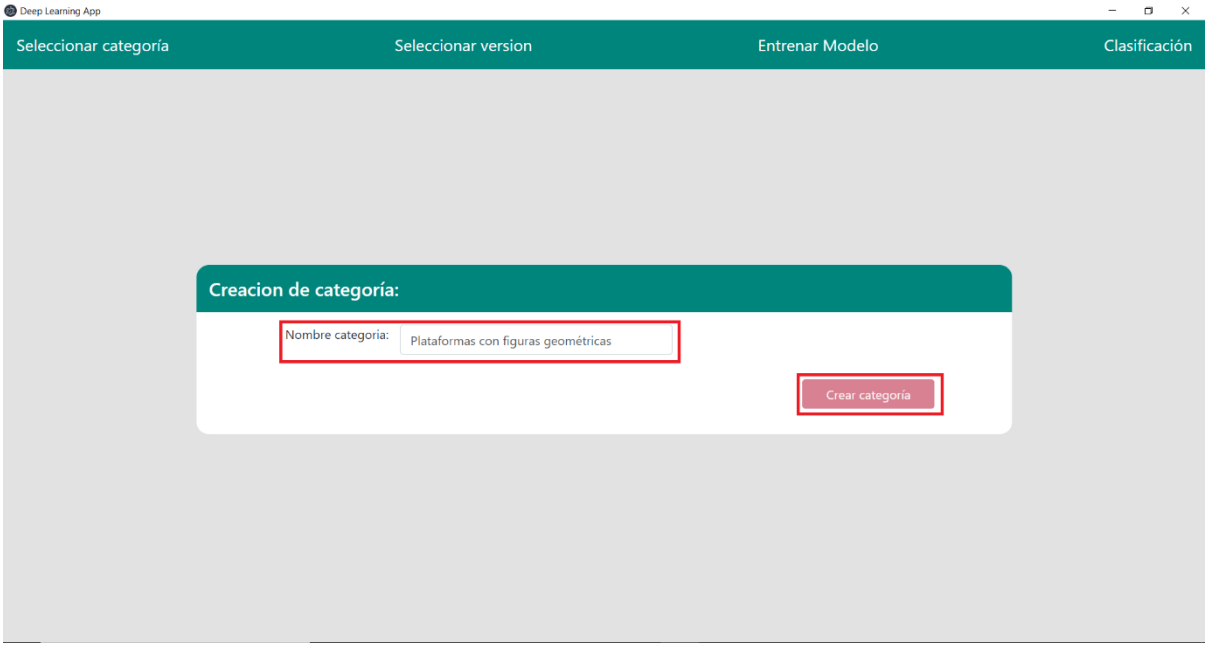

Figura 46: Formulario de creación de categoría

10.1.2 Gestionar una categoría

Una vez creada una categoría, la aplicación redirige al usuario otra vez a la vista principal donde se pueden ver todas las categorías.

Como ya se dispone de varias categorías, se puede seleccionar cualquiera de ellas, lo que permite dos nuevas funciones mediante los botones correspondientes, Figura 47:

- Borrar categoría: Elimina dicha categoría y todas las versiones asociadas a ella.

- Confirmar selección: Permite seleccionar dicha categoría para poder crear versiones o ver las versiones correspondientes. **Es importante confirmar la selección de la categoría antes de pasar a la vista de seleccionar versión.**

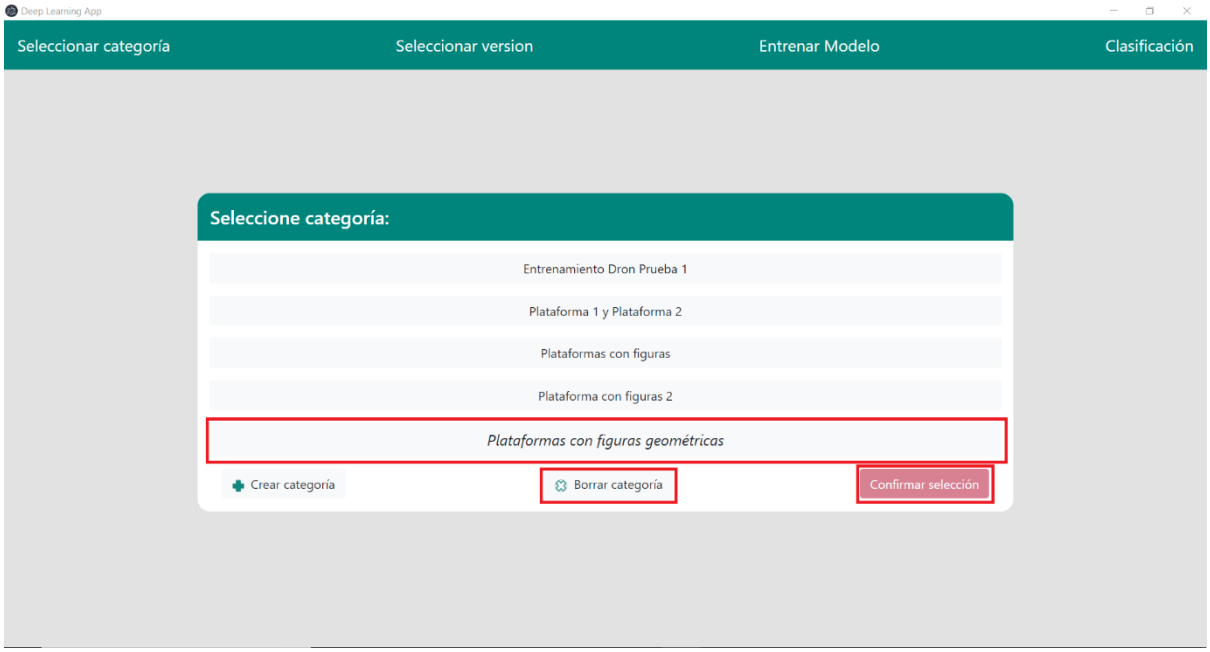

Figura 47: Seleccionar de selección de categoría

10.2 VISTA SELECCIONAR VERSIÓN

Al confirmar la selección de una categoría se puede pasar a la vista de las diferentes versiones asociadas a dicha categoría.

Si no seleccionamos una categoría y pasamos directamente a la vista "Seleccionar versión" la aplicación lanzará un mensaje de aviso y redirigirá a dicha vista, pero sin la posibilidad de crear ni ver ninguna versión.

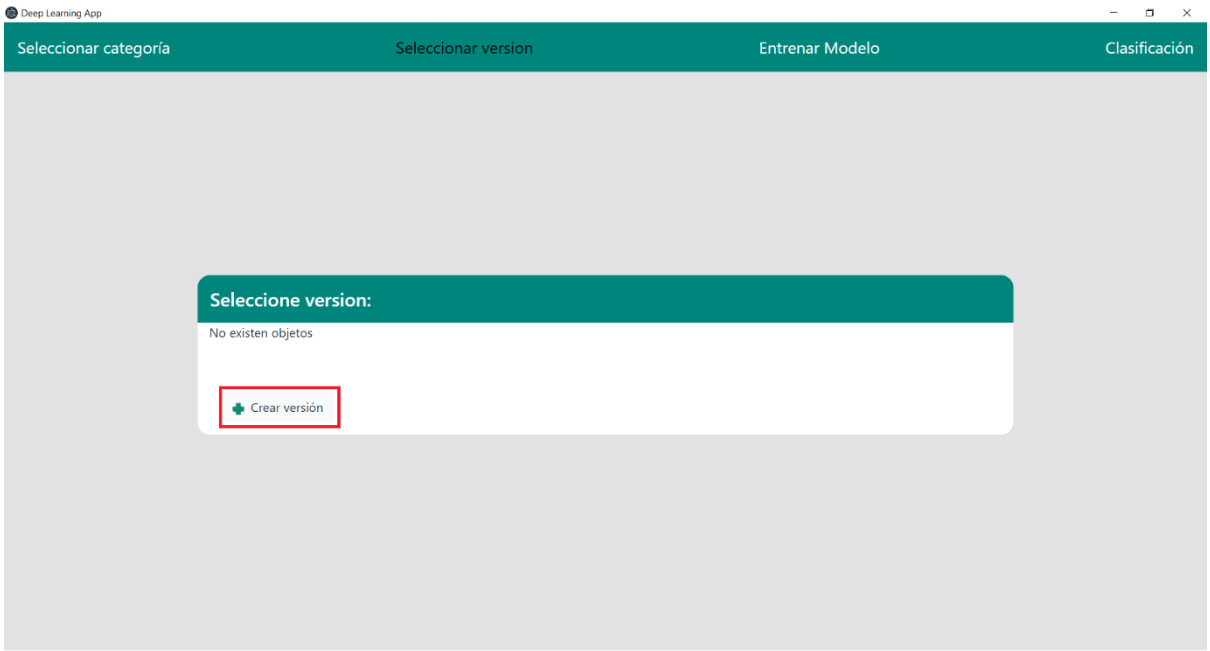

Figura 48: Vista de selección de versión

Al seleccionar la categoría creada en el apartado anterior y haciendo clic en el botón "Seleccionar versión" se puede ver una vista como la que aparece en la Figura 48.

Dicha vista carga automáticamente todas las versiones existentes. En el estado actual, como no se ha creado ninguna versión, no aparece ninguna.

10.2.1 Crear una versión

Además de cargar las versiones existentes en la categoría seleccionada se puede crear una nueva versión utilizando el botón “Crear versión”. Al pulsarlo se puede ver el formulario de la Figura 49.

Dicho formulario permite dar un nombre a la versión (modelo), cambiar las unidades de densidad y el ratio de aprendizaje del modelo; y también, añadir las clases que el modelo tiene que utilizar en el proceso de clasificación.

Para añadir una clase, pulsamos el botón “Añadir clase” lo que creará un nuevo campo que permitirá introducir una nueva clase. En la Figura 49 aparecen recuadradas en verde dos clases, esto es debido, a que anteriormente se había pulsado el botón “Añadir clase” dos veces. **Como mínimo hay que añadir dos clases a cada versión creada.**

Si se han añadido más clases de las deseadas, se puede eliminar la **última** clase creada pulsando el botón “Borrar clase”.

Si ya se ha rellenado todo el formulario correctamente, se confirma la creación de la versión utilizando el botón “Crear versión”.

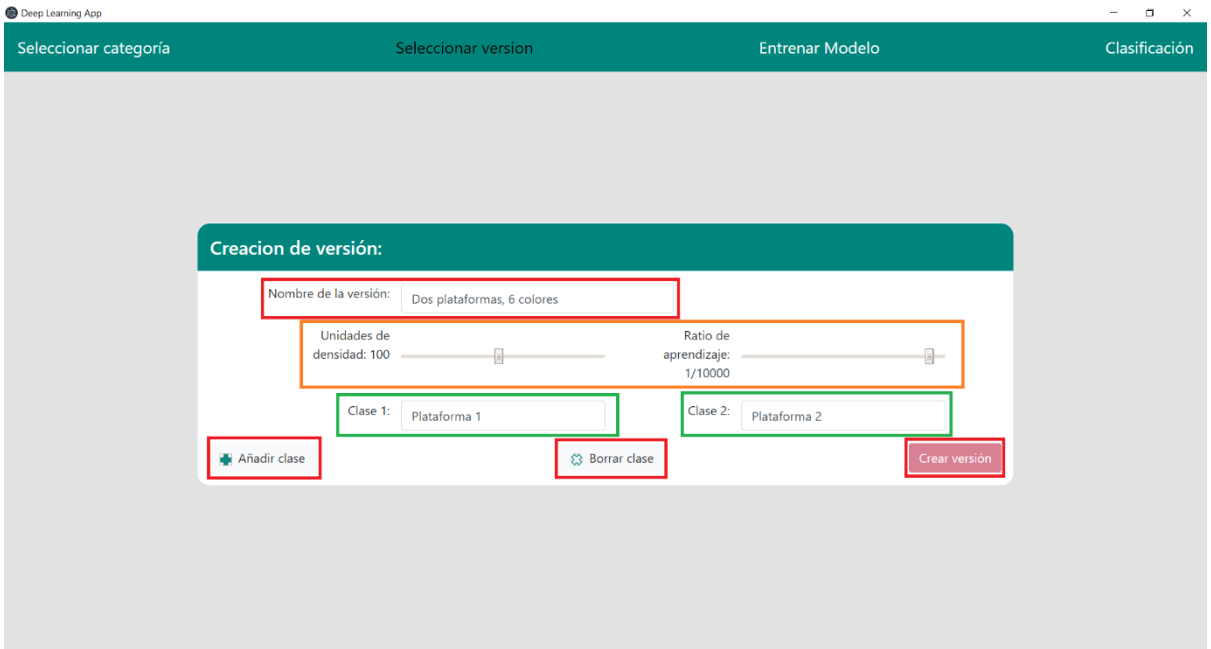

Figura 49: Crear una versión

10.2.2 Gestionar una versión

Una vez confirmada la creación de la versión, la aplicación redirige a la vista principal de las versiones, cargando dicha versión.
Al hacer clic en la versión (ver Figura 50) aparecen dos nuevas opciones, además de permitir crear otra versión:

- Borrar Versión: Borra la versión seleccionada de la categoría actual.

- Confirmar selección: Permite utilizar dicha versión (modelo) en las vistas “Entrenar modelo” y “Clasificación”.

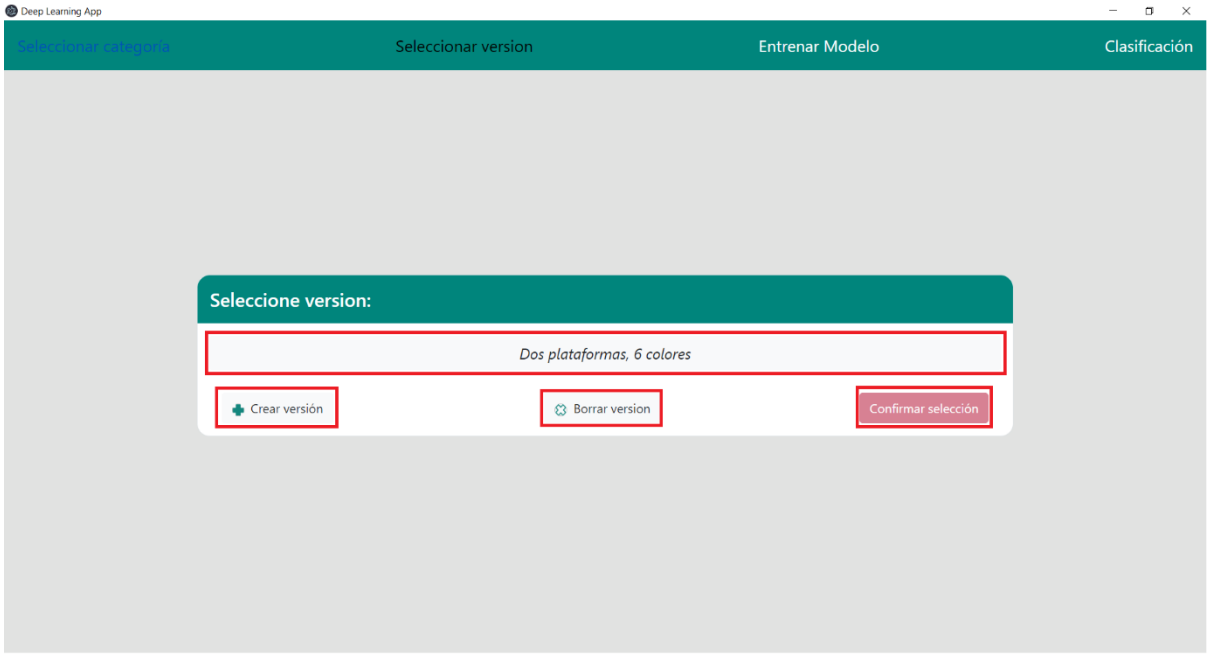

Figura 50: Seleccionar una versión

10.3 VISTA ENTRENAR MODELO

Al confirmar la selección de la versión en el apartado anterior, se puede pasar a entrenar un modelo haciendo clic en “Entrenar Modelo”.

Esta acción, redirige a la vista de entrenamiento de modelos como se puede observar en la Figura 51, la cual se puede dividir en dos zonas.

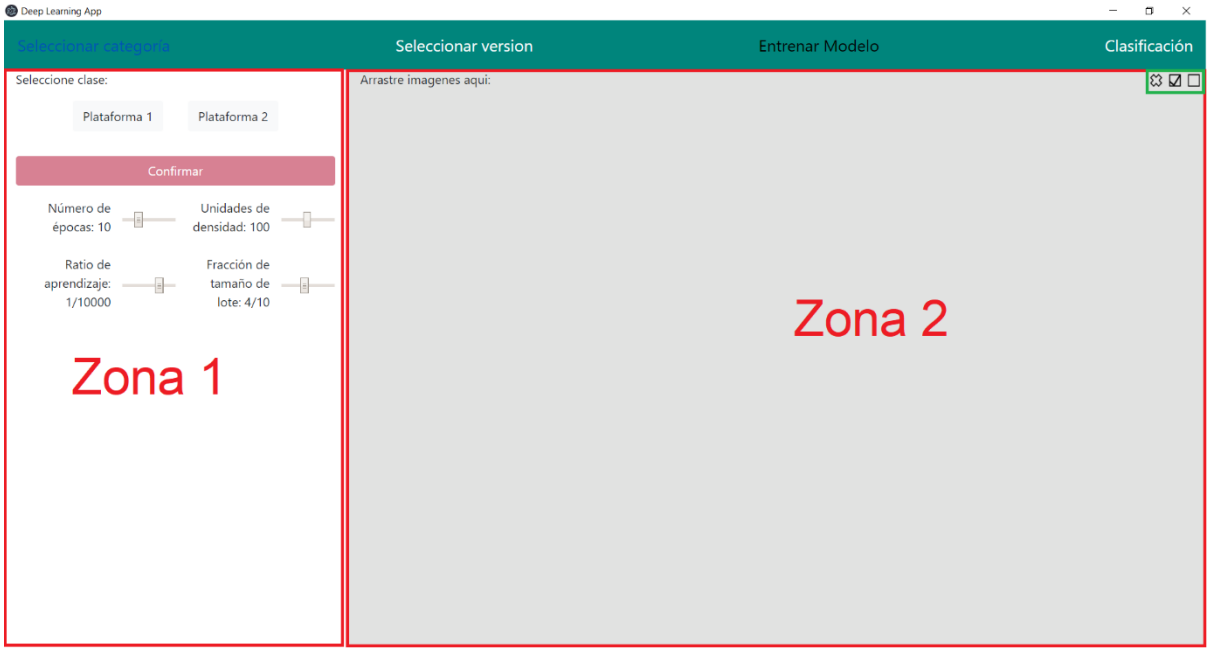

Figura 51: Vista de entrenar modelo

10.3.1 Añadir muestras para realizar un entrenamiento

En la “Zona 2” que aparece en la Figura 51 se puede “arrastrar” todas las muestras disponibles para realizar un entrenamiento.

Para realizar un entrenamiento más rápido y más cómodo es recomendable que el número de muestras por clase sea mayor que 5 y menor que cien. Hay que tener en cuenta que estamos en un entorno web y la carga de imágenes requiere cierto tiempo.

En esta zona, además, se pueden ver tres botones en la esquina superior derecha:

- El primer botón sirve para eliminar una o varias imágenes previamente seleccionadas por lo que no se van a utilizar para el entrenamiento.
- El segundo botón permite seleccionar todas las imágenes disponibles en la “Zona 2”.
- El tercer botón permite deseleccionar todas las imágenes seleccionadas de la “Zona 2”.

Estos botones son muy útiles a la hora de realizar entrenamientos.

10.3.2 Asignar imágenes a una clase y entrenar

Una vez añadidas las muestras necesarias, se pasa a la “Zona 1” de la Figura 51. En la cual, se cargan automáticamente las clases de la versión previamente seleccionada.

Además, aparece por defecto el botón de “Confirmar” que permite asociar imágenes de la “Zona 2” a una clase y los diferentes parámetros para los entrenamientos, los cuales se

recomienda mantener con su valor por defecto a no ser que se tengan conocimientos avanzados de redes neuronales.

Para realizar un entrenamiento completo se recomienda realizar los siguientes procesos:

1. Añadir un número de muestras comprendido entre cinco y cien de **una sola clase**.

2. Esperar hasta que la previsualización de todas las imágenes cargadas sea visible.

3. Como se han añadido todas las muestras de la misma clase, se utiliza el botón de la "Zona 2" para seleccionar todas las imágenes cargadas.

4. Se hace clic en el botón de la clase correspondiente de la "Zona 1".

5. Confirmar el entrenamiento pulsando el botón "Confirmar".

6. Automáticamente las imágenes pasan a tener un fondo gris con la clase asociada, como se puede ver en la Figura 53.

7. Repetir los pasos 1-6 con las muestras de todas las clases.

8. Una vez añadidas muestras de todas las clases disponibles y configurados los parámetros, se mostrará el botón "Entrenar".

9. Pulsar el botón "Entrenar" y esperar a que el entrenamiento finalice. En función de las muestras añadidas el entrenamiento puede tardar más o menos tiempo.

En la Figura 52 se pueden observar los primeros cinco pasos. La Figura 53, representa el estado de la aplicación una vez añadidas muestras de ambas clases y asociadas a sus correspondientes clases.

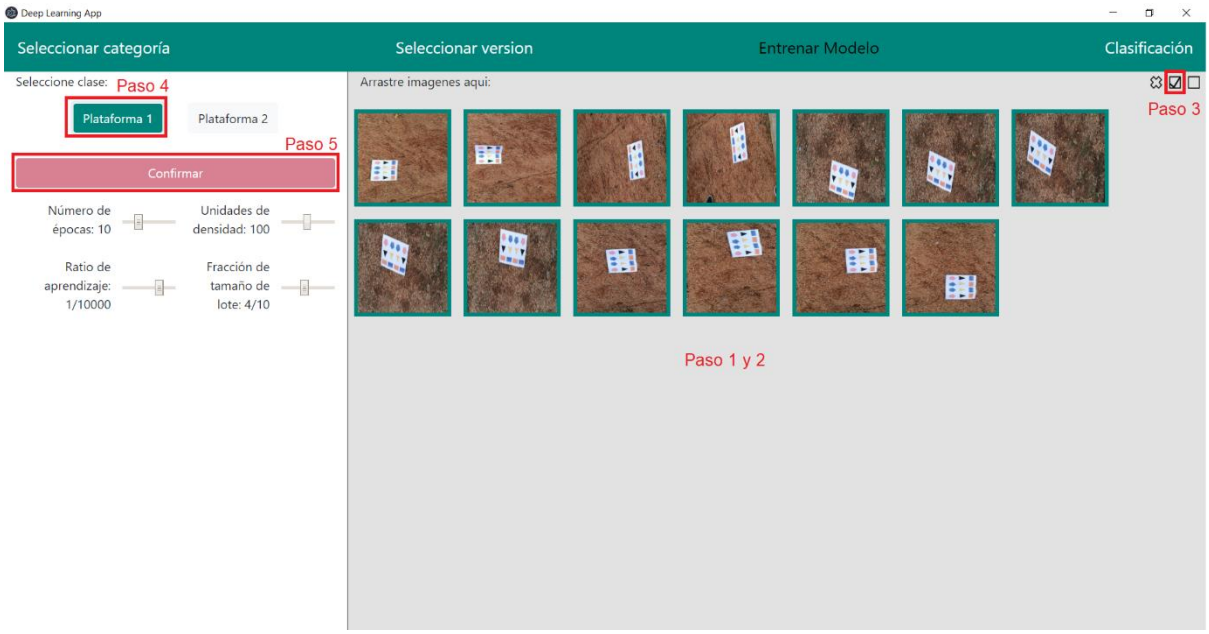

Figura 52: Entrenamiento, pasos 1-5

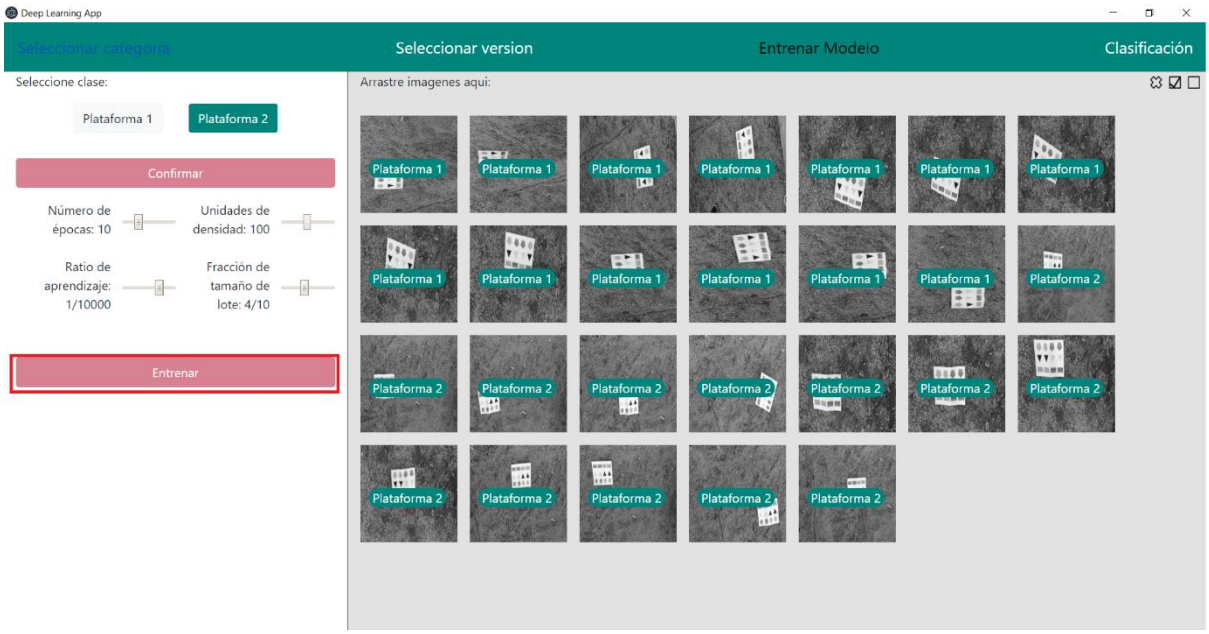

Figura 53: Muestras asociadas a sus clases

Asignadas todas las imágenes, el usuario pulsará sobre el botón “Entrenar” y esperará a que el entrenamiento finalice.

10.4 VISTA CLASIFICACIÓN

Una vez entrenado un modelo, se puede realizar una clasificación desde la vista “Clasificación”.

En la Figura 54 se puede observar que la vista de clasificación dispone de una zona para “arrastrar” imágenes (o si se hace un clic en dicha zona o alrededores aparecerá la selección de ficheros) y del botón “Clasificar” que utilizará la versión previamente seleccionada para realizar una clasificación de la imagen introducida.

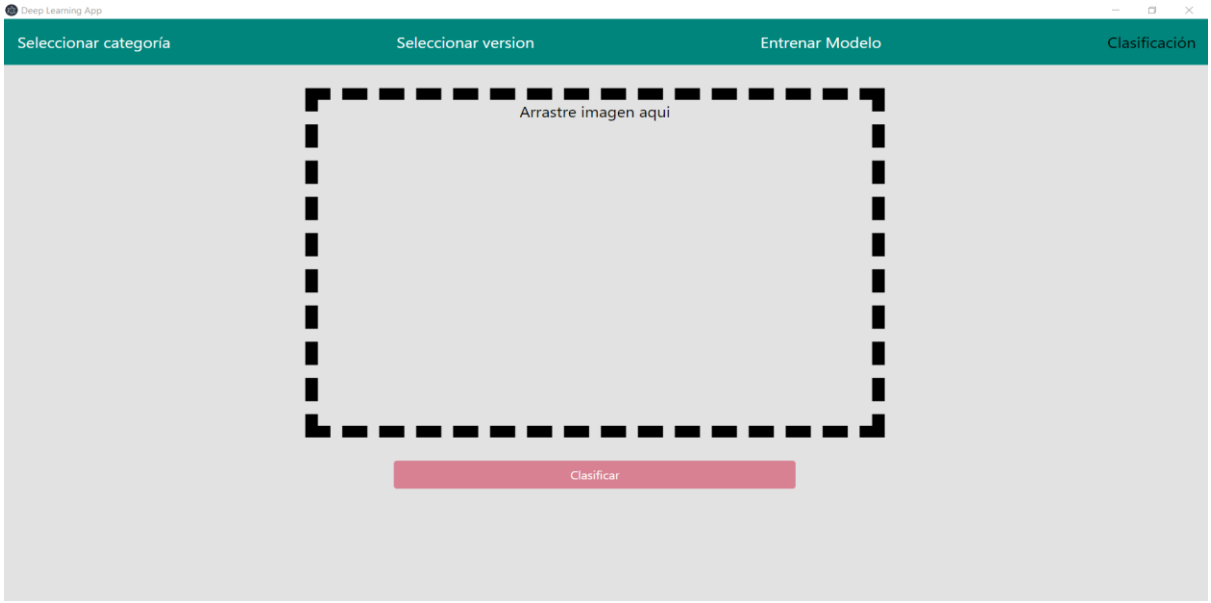

Figura 54: Vista de clasificación

En la Figura 55 se ha realizado una clasificación en base al entrenamiento realizado en el apartado anterior. Como se puede observar el resultado de la clasificación aparece debajo del botón “Clasificar”.

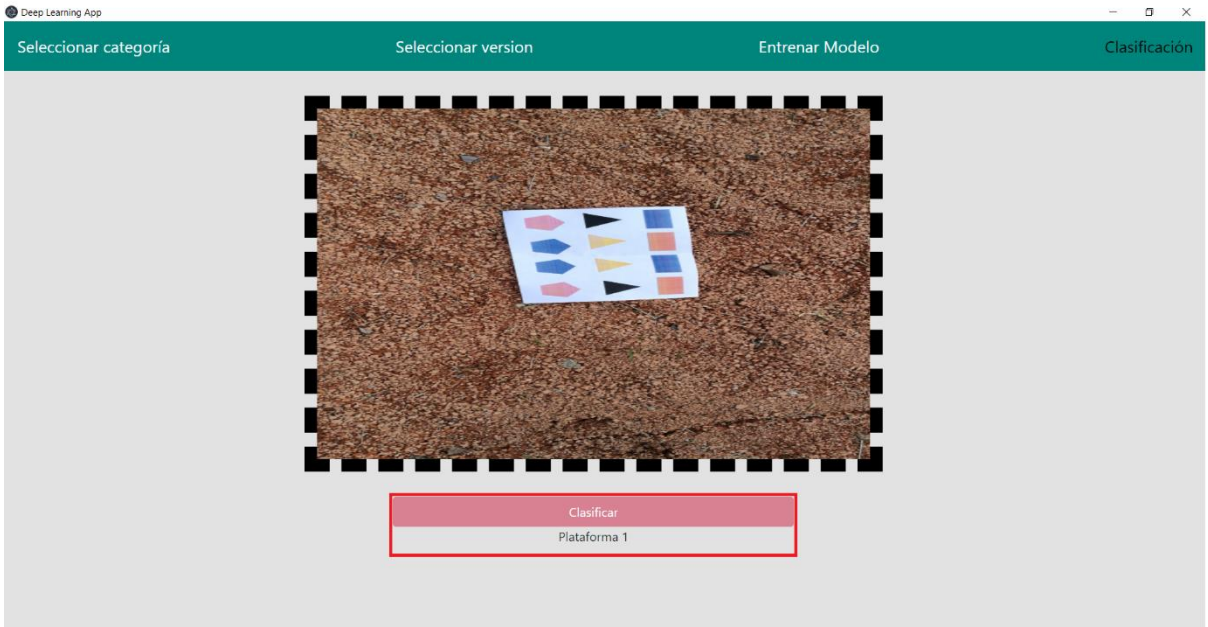

Figura 55: Clasificación de una imagen