



ESCUELA TÉCNICA SUPERIOR DE INGENIEROS  
INFORMÁTICOS

UNIVERSIDAD POLÍTÉCNICA DE MADRID

---

# Reconocimiento de landmarks visuales basado en Deep Learning + CNNs para la navegación autónoma en entornos mixtos de interior/exterior mediante mapas topológicos visuales

---

TRABAJO FIN DE MÁSTER  
MÁSTER UNIVERSITARIO EN INTELIGENCIA ARTIFICIAL

AUTOR: Amable José Valdés Cuervo  
TUTOR: Darío Maravall Gómez-Allende



# Agradecimientos

Quiero agradecerle a Zara, que me enseño el poder de tomar decisiones en mi vida para luchar por la vida que quiera tener. Sin ella nunca habría tomado la decisión de hacer este máster, que me ha enriquecido no solo como ingeniero, si no como persona.

Quiero agradecerle a mis padres por todo el apoyo y ayuda que me han brindado; no solamente económico (¡Que ya es mucho!), si no emocional. Otros padres podrían haber dicho “Si quieres una carrera trabaja y págatela”, pero ellos desde el principio me han animado a seguir estudiando para estar mejor preparado para el futuro.

Quiero agradecerle a Dario por la atención que ha puesto corrigiendo esta documentación, aportándome documentación tanto sobre mapas topológicos como de redes neuronales y guiándome a lo largo de todo este periodo final en el máster. Otros tutores podrían haberme ignorado, pero Dario siempre me enviaba correos comentándome cosas que podría mejorar o incluir en el proyecto. Ha sido muy agradable trabajar con él.

Podría incluir a mucha más gente (amigos, familiares...), pero ellos ya saben que les agradezco el haber estado ahí, y espero que sigan ahí en el futuro.

Por último, quiero agradecerle al lector que esté leyendo estas palabras, quien quiera que sea, simplemente por el hecho de tomarse su tiempo para leer este proyecto; espero que esta Tesis de Fin de Máster (TFM) sea de su agrado. Personalmente, he disfrutado haciendo este proyecto (¿Quién no se divierte creando una red neuronal? ¡Ha sido muy entretenido!) y he intentado explicarlo todo de la manera más clara y concisa posible.



# Abstract

El estudio de un mapa topológico visual para la navegación de un robot móvil tiene muchas aplicaciones en la actualidad; desde sistemas de ayuda para gente con deficiencia visual hasta la navegación autónoma de robots para la automatización de tareas en una empresa.

En esta tesis diseñamos y analizamos un mapa topológico de interiores y exteriores seleccionando landmarks visuales característicos del entorno. Asimismo, creamos un dataset para entrenar a un clasificador para que identifique estos lugares.

Mediante las técnicas de aprendizaje profundo o *deep learning* hemos creado una red de neuronas artificiales de manera rápida y simple que mediante capas convolucionales y otras técnicas reconoce estos landmark visuales con una precisión elevada que ronda el 90 % de tasa de aciertos. Para llegar a estos resultados hemos realizado pruebas de validación y de test que avalan el correcto funcionamiento de la red, que discutiremos en diversos apartados de esta misma tesis.

Para finalizar, hablaremos de los problemas con los que nos hemos encontrado tanto a lo largo de la realización de este proyecto como en los resultados finales y discutiremos las posibles mejoras de este enfoque mediante otras técnicas.



# Índice general

<b>Agradecimientos</b>	<b>III</b>
<b>Abstract</b>	<b>V</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Antecedentes . . . . .	1
1.2. Objetivos . . . . .	2
<b>2. Descripción del entorno experimental de navegación</b>	<b>5</b>
2.1. Selección del entorno . . . . .	5
2.2. Mapa topológico . . . . .	6
2.3. Landmarks seleccionados . . . . .	7
2.3.1. El dilema de concreción versus generalidad . . . . .	7
2.3.2. Landmarks en detalle . . . . .	8
2.4. Navegación . . . . .	13
<b>3. Generación del dataset de imágenes de landmarks para el entrenamiento y evaluación de los reconocedores</b>	<b>15</b>
3.1. Definición del dataset . . . . .	15
3.2. Toma de imágenes . . . . .	16
3.3. Tratamiento de las imágenes . . . . .	17
3.4. Contenido del Dataset . . . . .	17
3.4.1. Training . . . . .	17
3.4.2. Validation . . . . .	18
<b>4. Clasificador: Deep Learning + CNNs</b>	<b>19</b>
4.1. Implementación . . . . .	19
4.1.1. Herramientas . . . . .	19
4.1.2. Sistema . . . . .	20
4.1.3. Arquitectura de la red . . . . .	20
4.2. Técnicas usadas . . . . .	20
4.2.1. CNNs - Convolutional Neural Networks . . . . .	21
4.2.2. Data augmentation . . . . .	22
4.2.3. Max pooling . . . . .	24
4.2.4. Dropout . . . . .	24

4.2.5. Batch Normalization . . . . .	25
4.2.6. Activaciones ELU . . . . .	25
4.3. Entrenamiento . . . . .	25
<b>5. Validation: test estáticos</b>	<b>29</b>
5.1. Jupyter notebook . . . . .	29
5.2. Conjunto de datos de validación . . . . .	29
5.3. Proceso de validación . . . . .	30
5.4. Resultados . . . . .	30
<b>6. Test: tests dinámicos</b>	<b>33</b>
6.1. Jupyter notebook . . . . .	33
6.2. Conjunto de datos de test . . . . .	33
6.3. Proceso de test . . . . .	33
6.4. Resultados . . . . .	35
<b>7. Conclusiones y líneas futuras de trabajo</b>	<b>37</b>
7.1. Conclusiones . . . . .	37
7.2. Otras lineas de trabajo . . . . .	38
7.2.1. Bolsa de palabras visuales . . . . .	38
7.2.2. R-CNN o YoLo . . . . .	39
<b>A. Training completo</b>	<b>41</b>
<b>Bibliografía</b>	<b>45</b>

# Índice de figuras

1.1.	Mapa estudiado en la asignatura de robots autónomos . . . . .	1
2.1.	Mapa de la zona de estudio . . . . .	5
2.2.	Mapa topológico del proyecto . . . . .	6
2.3.	Áreas de estudio de cada nodo . . . . .	8
2.4.	Ejemplos de imágenes pertenecientes al nodo 1, el pasillo de las habitaciones . . . . .	9
2.5.	Ejemplos de imágenes pertenecientes al nodo 2, las escaleras del hall .	10
2.6.	Ejemplos de imágenes pertenecientes al nodo 3, el comedor . . . . .	10
2.7.	Ejemplos de imágenes pertenecientes al nodo 4, el edificio de las habitaciones . . . . .	11
2.8.	Ejemplos de imágenes pertenecientes al nodo 5, el parking . . . . .	12
2.9.	Ejemplos de imágenes pertenecientes al nodo 6, la biblioteca . . . . .	12
2.10.	Ejemplos de imágenes pertenecientes al nodo 7, el gimnasio . . . . .	13
3.1.	Dataset para Redes Neuronales . . . . .	15
3.2.	División del dataset para este proyecto . . . . .	18
4.1.	Funcionamiento de las capas convolucionales . . . . .	22
4.2.	Generación de una nueva imagen invertida (imagen espejo, “flip”) a partir de otra imagen. . . . .	23
4.3.	Generación de nuevas imágenes con giros a partir de una imagen original. . . . .	23
4.4.	Ejemplo del funcionamiento de las capas Max Pooling 2D . . . . .	24
4.5.	Evolución del entrenamiento en cada epoch mostrando el <i>Loss</i> y la <i>Accuracy</i> . . . . .	27
4.6.	Evolución de la <i>Accuracy</i> en cada epoch . . . . .	27
4.7.	Evolución del valor <i>Loss</i> en cada epoch . . . . .	28
5.1.	Output en el entrenamiento de una red neuronal con Keras . . . . .	30
5.2.	Output del método <code>.evaluate_generator</code> . . . . .	30
5.3.	Output del Jupyter Notebook con los resultados de nuestra red neuronal mediante los métodos de kera y los scripts implementados . . . . .	31
5.4.	Matriz de confusión . . . . .	31
5.5.	Matriz de confusión normalizada . . . . .	32

6.1.	Output de las pruebas test . . . . .	35
6.2.	Superposición de nodos entre el nodo 3 (Building - Dinning room) y el nodo 5 (Parking) . . . . .	36
7.1.	Superposición de tres nodos en las pruebas test. La red neuronal duda entre cual de los tres clasificar el frame . . . . .	38
7.2.	Funcionamiento de la bolsa de palabras (Imágenes pertenecientes a la asignatura de <i>Visión por computador</i> de este mismo máster) . . . . .	39
7.3.	Representación de lo que veríamos con una red R-CNN o YoLo . . . . .	40

# Índice de tablas

4.1. Arquitectura de la red neuronal . . . . .	21
4.2. Entrenamiento de la red . . . . .	26
A.1. Datos del entrenamiento completo de la red . . . . .	41
A.2. Datos del entrenamiento completo de la red . . . . .	42
A.3. Datos del entrenamiento completo de la red . . . . .	43



# Índice de algoritmos

- |    |                                 |    |
|----|---------------------------------|----|
| 1. | <i>dinamic_test(name_video)</i> | 34 |
|----|---------------------------------|----|



# Capítulo 1

## Introducción

### 1.1. Antecedentes

Parto de un trabajo anterior realizado en la asignatura de robots autónomos de este mismo máster en el que mis compañeros y yo realizamos un diseño de un mapa topológico y entrenamos un K-NN para identificar la zona donde se encontraba.

El mapa topológico que diseñamos representaba el segundo piso del bloque 2-3; el camino desde las escaleras del bloque 2 hasta el aula 3202. Dicho mapa topológico puede verse en la Figura 1.1.

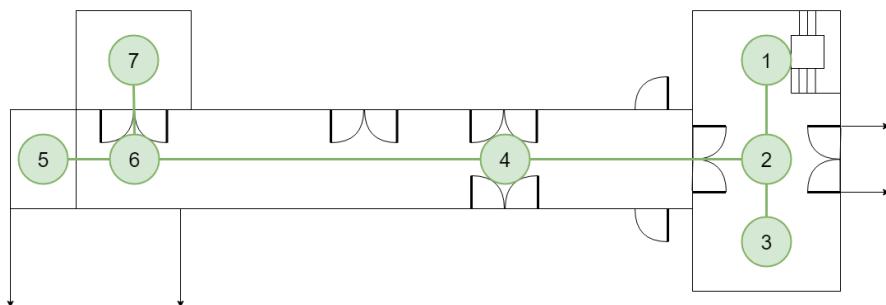


Figura 1.1: Mapa estudiado en la asignatura de robots autónomos

Cada nodo representado en el mapa topológico de la Figura 1.1 es un landmark visual representativo de cada zona del pasillo y los arcos indican desde qué nodo se puede viajar hasta cuales nodos.

*El dataset:* Entrenamos al K-NN con un dataset que constaba de unas 4200 imágenes, de las que 4090 fueron usadas para entrenamiento y 110 para validación, y un vídeo para test. Esto se traduce en que para cada uno de estos nodos se tomaron unas 600 imágenes de las que alrededor de unas 590 sirvieron para entrenamiento y alrededor de unas 10 para validación.

Para usar estas imágenes para entrenar al K-NN convertimos esas imágenes en color a imágenes en escala de grises y, a su vez, esas imágenes en escala de grises a

histogramas. Mediante el estudio de estos histogramas el K-NN podía predecir, dada una nueva foto que nunca hubiera visto, cuál era la imagen que más se le parecía de entre todas las imágenes de entrenamiento.

- Dataset, 4200 imágenes y 1 vídeo.
  - Training, 4090 histogramas de imágenes en escala de grises.
  - Validation, 110 histogramas de imágenes en escala de grises.
  - Test, histogramas de los frames de 1 único vídeo con el recorrido a estudiar.

*Conclusiones:* Los resultados de esta aproximación fueron los esperados; buenos resultados en validación y malos resultados en test.

- En validación el K-NN tuvo una tasa de aciertos del 99 %, acertando 109 de las 110 imágenes.
- En test el K-NN tuvo una tasa de aciertos del 46 %, acertando 552 frames de los 1192 frames del vídeo.

Con un K-NN, que es un clasificador sencillo, los resultados en la validación son muy buenos, pero es un clasificador que no generaliza ya que simplemente se fija en los niveles del histograma de la imagen y debido a ello los resultados en test son muy malos como se podía esperar, no llegando a acertar el nodo en el que se encuentra ni la mitad de las veces. Además, hay que añadir que el dataset es pequeño; con unas 600 imágenes por nodo y tan solo 10 imágenes para la validación es normal que los resultados no sean de mejor calidad.

## 1.2. Objetivos

Con este proyecto espero conseguir una mejora significativa de los resultados obtenidos en el proyecto pasado usando técnicas novedosas en el campo de la visión por computador como es el Deep Learning junto al uso de redes convolucionales o CNNs.

Para conseguir esto se van ha de realizar una serie de tareas:

- **Nuevo mapa topológico:** Se diseñará un nuevo mapa topológico con entornos mixtos de interior y exterior con landmarks bien diferenciados.
- **Generación de un nuevo dataset:** Se tomarán imágenes de los landmark visuales característicos del mapa topológico.
- **Deep Learning:** Se modelizará una red neuronal que será entrenada con el nuevo dataset.

- **Estudio de los resultados:** Se le realizarán pruebas a la red neuronal con los datos de validación y test. Se estudiarán estos resultados centrándose en la exactitud de sus predicciones y sus errores, comprobando así la eficiencia y rendimiento de la red neuronal.



# Capítulo 2

## Descripción del entorno experimental de navegación

### 2.1. Selección del entorno

En este proyecto voy a trabajar sobre un entorno distinto al presentado en los antecedentes en 1.1. El anterior entorno era completamente de interior y he decidido ampliarlo a un entorno mixto de interior/exterior, siendo este nuevo entorno los terrenos de la residencia de estudiantes donde he residido durante el periodo lectivo. Dicho entorno puede verse representado en la Figura 2.1.

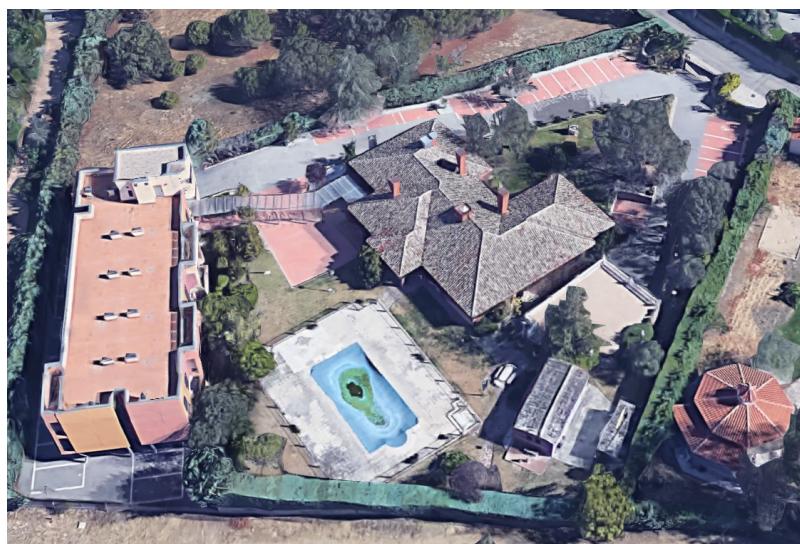


Figura 2.1: Mapa de la zona de estudio

Las razones de esta ampliación son varias:

- **Texturas y formas:** En el entorno de los antecedentes prevalecen solo un par de texturas y/o colores: el suelo, negro o beige; las paredes, siempre blancas y con una franja gris o negra; algún marrón para elementos de madera como

las puertas. Con este nuevo entorno tenemos mucha más variedad de texturas: paredes pintadas de distintos colores; suelos de ladrillo, piedra, madera o distintos colores; edificios con texturas características.

Esto dificulta el entrenamiento de la red, ya que esta ha de generalizar muchas más texturas y elementos, pero da mucho juego para la identificación de los landmark y enriquece el entorno de experimentación.

- **Iluminación:** Al trabajar en un entorno mixto de interiores/exteriores la iluminación en muchos landmark será distinta dependiendo de las condiciones climáticas (sol, nubes, lluvia), no como ocurría en el entorno de los antecendentes donde solo nos preocupábamos por la iluminación para comparar los histogramas y donde no había tanta variedad.

Esto dará ventajas y desventajas a la hora de realizar el dataset: como desventaja, hay que asegurarse de tomar imágenes en varias horas del día con distinta iluminación y condiciones climáticas; como ventaja, la red generalizará mejor con imágenes con distinta iluminación, lo que nos dará mejores resultados.

## 2.2. Mapa topológico

He analizado el entorno experimental de la misma manera que se realiza en [3]; buscando lugares representativos que podrían servir como nodos clave para definir los landmarks visuales del mapa topológico y estudiando como se puede navegar desde un landmark a otro.

Tras el análisis inicial he llegado a la confección del mapa topológico que puede observarse en la Figura 2.2.

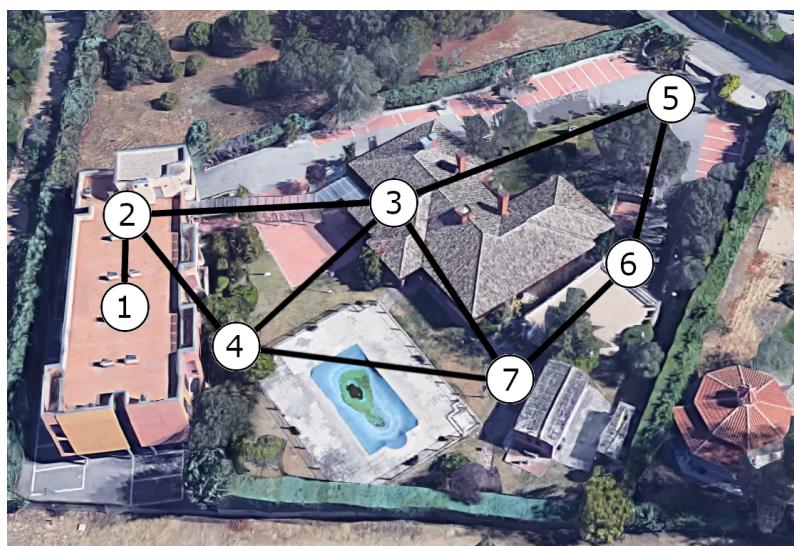


Figura 2.2: Mapa topológico del proyecto

En dicho mapa puede observarse la relación entre los distintos nodos y como desde un nodo se llega a otros a través de la conexión de sus arcos. Más adelante en el apartado 2.3.2 explicaré cada nodo en detalle.

## 2.3. Landmarks seleccionados

### 2.3.1. El dilema de concreción versus generalidad

La selección de los landmarks y sus respectivas imágenes es una etapa crucial en la generación manual de un mapa topológico visual basado en grafos visuales ya que plantea un dilema crítico entre la concreción o especialización por una parte y la generalidad por otra de las imágenes asociadas a los landmarks del mapa.

Se puede afirmar que los landmarks “generalistas” tienen la ventaja de una mayor simplicidad y ergonomía antropomórfica de los correspondientes mapas topológicos visuales (en el sentido de ser más intuitivos o perceptualmente comprensibles para un usuario humano, lo cual facilita el desarrollo del diseño y testeo de los mapas y de las pruebas del navegador, así como la posible interacción humano-robot en un entorno de navegación física del robot con intervención o cooperación humana).

No obstante, frente a la citada ventaja “antropomórfica” de los landmarks “generalistas”, éstos también plantean un reto mayor en el desarrollo del módulo de reconocimiento ya que las imágenes asociadas a los landmarks “generalistas” son más complejas que las imágenes asociadas a landmarks “particularistas”. Se puede afirmar que los landmarks generalistas requieren reconocedores de elevada capacidad de generalización, contrariamente a los reconocedores de landmarks particularistas, mucho más especializados.

En [2] puede observarse que se han definido los landmarks mediante la selección de objetos concretos y bien diferenciados. Este suele ser el procedimiento más empleado: por ejemplo, en [3] se seleccionaron una puerta o un extintor. En el presente trabajo no se ha optado por este habitual enfoque particularista de selección de los landmarks centrados en objetos concretos y bien diferenciados, sino que hemos optado por un enfoque más generalistas.

El entorno de los antecedentes o de [2, 3] son entornos pequeños; una habitación o pasillos de unos metros de distancia como mucho. En el entorno que hemos definido en este proyecto tenemos media hectárea de terreno de estudio.

Debido al tamaño del entorno las imágenes que tomaremos para nuestro dataset serán imágenes muy genéricas que contendrán muchos objetos y estructuras que se repetirán a lo largo de todos los nodos tales como puertas, carteles, escaleras... Además de tener presente el hecho de que muchos de nuestros landmark son objetos compuestos por otros objetos, como pueden ser los edificios que clasificaremos que están definidos por ventanas, puertas, chimeneas, etc.

Este entorno podría dar problemas a clasificadores que se centran en buscar objetos concretos, y es por eso por lo que para este proyecto nos vamos a apoyar en redes neuronales y, en especial, en las redes convolucionales donde la generalización en imágenes es uno de los aspectos claves usados para la correcta clasificación.

De esta manera tomaremos imágenes desde distintos ángulos de los escenarios que queremos clasificar y dejaremos que la red neuronal encuentre patrones comunes en la imagen para poder llegar a una predicción acertada del lugar donde se encuentra. Para esto hemos definido una serie de zonas donde tomaremos imágenes de cada landmark. Estas áreas definidas se pueden ver en la Figura 2.3.

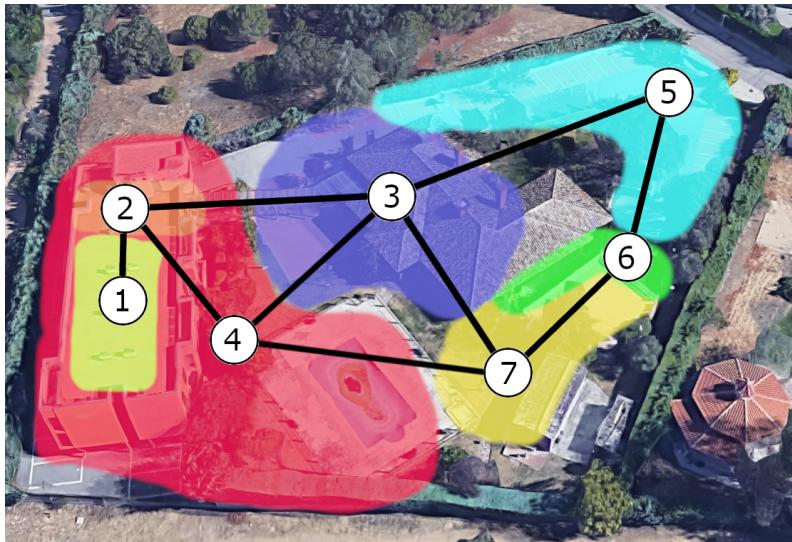


Figura 2.3: Áreas de estudio de cada nodo

Las imágenes del dataset serán tomadas desde las áreas presentadas en la Figura 2.3 enfocadas en un landmark más genérico que puede estar compuesto por muchos objetos que incluso pueden o no estar en la imagen. Esto quiere decir que seguiremos teniendo la relación "1 landmark = 1 nodo", pero el landmark será más complejo.

En el siguiente apartado 2.3.2 explicaré en detalle cada uno de los nodos seleccionados y su landmark asociado, definiendo que objetos aparecerán en la imagen y cómo esperamos que la red generalice para poder identificar el nodo.

En resumen, en este trabajo hemos decidido investigar la problemática asociada al uso de landmarks generalistas y por ese motivo nos hemos decantado hacia la experimentación con los métodos basados en redes neuronales convolucionales y aprendizaje profundo, en principio con mayor capacidad de generalización y reconocimiento complejo que otras técnicas de naturaleza más especializada.

### 2.3.2. Landmarks en detalle

A continuación explicaré en detalle cada uno de los nodos definidos.

- **Nodo 1 - Pasillo de las habitaciones**

- *Descripción:* Dentro del edificio donde se encuentran las habitaciones de los estudiantes este nodo comprende todo el pasillo del edificio.
- *Conexiones:* Desde este nodo se puede ir al nodo 2, las escaleras del hall.

- *Landmark*: El pasillo.
- *Generalización*: La red deberá generalizar usando: el pasillo, las paredes distintivas de color blanco y verde, las puertas de las habitaciones y las ventanas que dan al exterior.
- *Ejemplos*: Se pueden ver algunos ejemplos de este nodo en la Figura 2.4

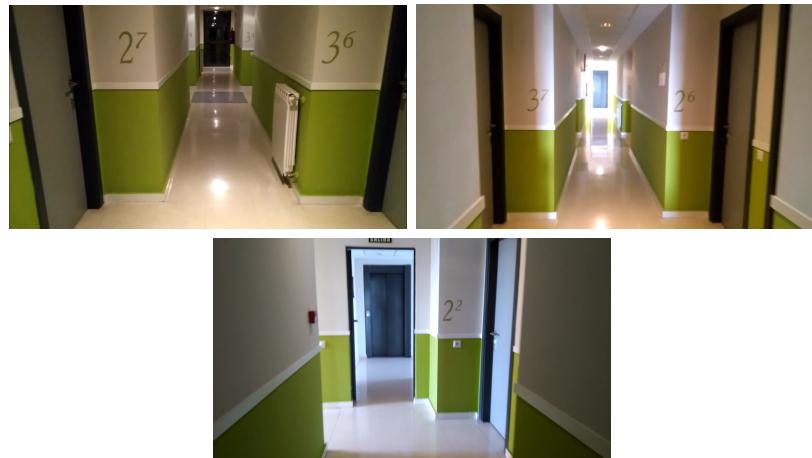


Figura 2.4: Ejemplos de imágenes pertenecientes al nodo 1, el pasillo de las habitaciones

#### ■ Nodo 2 - Escaleras del hall

- *Descripción*: Dentro del edificio donde se encuentran las habitaciones de los estudiantes este nodo comprende el hall del edificio.
- *Conexiones*: Desde este nodo se puede ir a: el nodo 1, el pasillo de las habitaciones; el nodo 4, el edificio de las habitaciones; y el nodo 3, el comedor.
- *Landmark*: La escalera.
- *Generalización*: La red deberá generalizar usando: el hall, las escaleras del hall, las paredes distintivas de color blanco, la puerta del ascensor y el dispensador de agua.
- *Ejemplos*: Se pueden ver algunos ejemplos de este nodo en la Figura 2.5

#### ■ Nodo 3 - Comedor

- *Descripción*: En el exterior se encuentra la fachada del edificio donde se encuentra el comedor y las cocinas de la residencia.
- *Conexiones*: Desde este nodo se puede ir a: el nodo 2, el hall; el nodo 4, la fachada del edificio de las habitaciones; el nodo 7, el gimnasio; y el nodo 5, el parking.

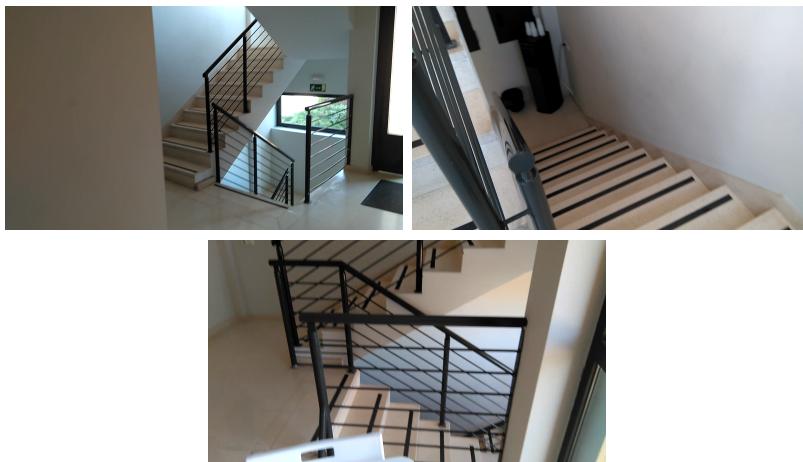


Figura 2.5: Ejemplos de imágenes pertenecientes al nodo 2, las escaleras del hall

- *Landmark:* La fachada del comedor.
- *Generalización:* La red deberá generalizar usando: el edificio donde se encuentra el comedor y las cocinas, que a su vez se encuentra compuesto por ventanas, puertas y la chimenea; los cubos de basura; sillas y mesas de exterior; las texturas del suelo y los ladrillos del edificio; algunos arboles de color característico.
- *Ejemplos:* Se pueden ver algunos ejemplos de este nodo en la Figura 2.6



Figura 2.6: Ejemplos de imágenes pertenecientes al nodo 3, el comedor

#### ■ Nodo 4 - Edificio de las habitaciones

- *Descripción:* En el exterior se encuentra la fachada del edificio donde se encuentran las habitaciones de los estudiantes.
- *Conexiones:* Desde este nodo se puede ir a: el nodo 3, el comedor; el nodo 2, las escaleras del hall; y el nodo 7, el gimnasio. El nodo 1 y 2 están dentro de este edificio.

- *Landmark:* La fachada de las habitaciones.
- *Generalización:* La red deberá generalizar usando la fachada del edificio donde se encuentran las habitaciones de los estudiantes, que a su vez se encuentra compuesto por ventanas, puertas, la chimenea, el cartel que indica que se trata del edificio de las habitaciones y las formas cuadradas o rectangulares pertenecientes a la arquitectura del edificio o los colores con los que ha sido pintada la fachada.
- *Ejemplos:* Se pueden ver algunos ejemplos de este nodo en la Figura 2.7



Figura 2.7: Ejemplos de imágenes pertenecientes al nodo 4, el edificio de las habitaciones

#### ■ Nodo 5 - Parking

- *Descripción:* En el exterior se encuentra el parking desde el que se consigue acceso a la calle y al exterior de la residencia.
- *Conexiones:* Desde este nodo se puede ir a: el nodo 3, el comedor; y el nodo 6, la biblioteca.
- *Landmark:* La puerta del parking
- *Generalización:* La red deberá generalizar usando las zonas de parking donde se pueden aparcar los coches, las texturas de piedra o ladrillo del suelo y paredes, los coches y la puerta negra de metal que da acceso al exterior de la residencia.
- *Ejemplos:* Se pueden ver algunos ejemplos de este nodo en la Figura 2.8.

#### ■ Nodo 6 - Biblioteca

- *Descripción:* En el exterior se encuentra la fachada de la biblioteca o zona de estudios de la residencia.

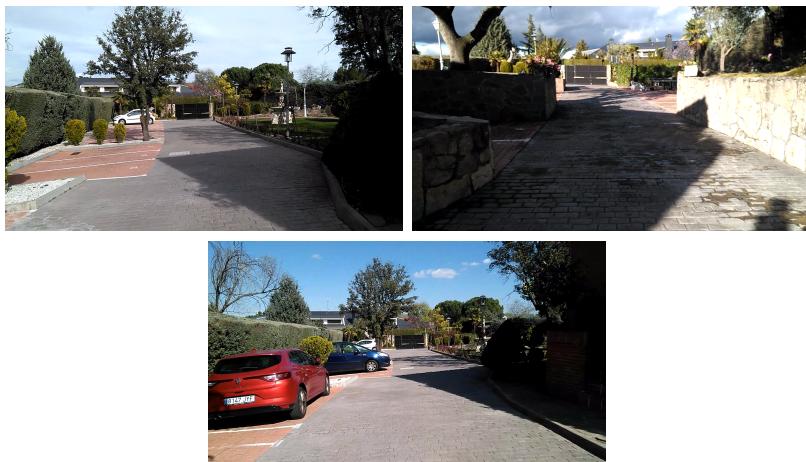


Figura 2.8: Ejemplos de imágenes pertenecientes al nodo 5, el parking

- *Conexiones:* Desde este nodo se puede ir a: el nodo 5, el parking; y el nodo 7, el gimnasio.
- *Landmark:* La fachada de la biblioteca.
- *Generalización:* La red deberá generalizar usando las paredes de piedra que rodean al edificio, las grandes ventanas y puerta de cristal y el cartel que indica que se trata de la biblioteca.
- *Ejemplos:* Se pueden ver algunos ejemplos de este nodo en la Figura 2.9



Figura 2.9: Ejemplos de imágenes pertenecientes al nodo 6, la biblioteca

#### ■ Nodo 7 - Gimnasio

- *Descripción:* En el exterior se encuentra la fachada del gimnasio de la residencia
- *Conexiones:* Desde este nodo se puede ir a: el nodo 3, el comedor; el nodo 4, el edificio de las habitaciones; y el nodo 6, la biblioteca.

- *Landmark:* La fachada del gimnasio.
- *Generalización:* La red deberá generalizar usando la fachada del edificio del gimnasio, las texturas y colores del edificio, el cartel que indica que se trata del gimnasio, los alrededores del gimnasio como la zona de ocio o la mesa de piedra.
- *Ejemplos:* Se pueden ver algunos ejemplos de este nodo en la Figura 2.10



Figura 2.10: Ejemplos de imágenes pertenecientes al nodo 7, el gimnasio

## 2.4. Navegación

En este proyecto de tesis de fin de máster no contamos con ningún robot físico por lo que la navegación no será una de las prioridades que tendremos que tener en mente para esta tesis.

No obstante este proyecto iría enfocado con el objetivo de que al finalizarlo un robot físico pudiese moverse por el mapa topológico diseñado gracias a nuestra red neuronal, y para ello debemos especificar algunos apartados importantes sobre la navegación en el mapa:

- *Robot físico:* En los terrenos de interior un robot sencillo (con, por ejemplo, un par de ruedas y un sistema odométrico) podría moverse sin ninguna dificultad, pero en este proyecto estamos trabajando en entornos mixtos de interior/exterior y eso complica la navegación pues en el exterior el terreno puede ser más complejo que unas losas de piedra o madera (como podemos encontrarnos en interiores), encontrándonos con distintos terrenos como piedra, tierra, hierba o ladrillo con distinta dificultad de movimiento en cada uno y, por no añadir, que el robot también tendría que enfrentarse a la dificultad de subir escaleras.

Una de las soluciones a este problema sería usar robots polimorfos que pudiesen enfrentarse a estos terrenos abruptos.

Otra posible solución sería usar un UAV (“Unmanned Aerial Vehicle”, un Dron). De esta manera podríamos olvidarnos del problema de los terrenos abruptos pues nuestro robot se movería por el aire.

- *Maniobrabilidad:* Los arcos de los nodos indican desde que nodo se puede ir a cual nodo, pero además (y aunque no se haya añadido en esta tesis) deberían contener información de como realizar las maniobras para viajar entre nodos. Estas maniobras deberían ser lo más simples y robustas posibles para el correcto funcionamiento del robot y para una navegación sin colisiones contra objetos del entorno.

También quiero remarcar que añadir en los arcos el conocimiento de las maniobras no es imprescindible; siempre se puede realizar la navegación con algoritmos reactivos, como el algoritmo bug. También se podría, una vez realizado un mapa interno simbólico de cada nodo, usar algoritmos de resolución de rutas como el A\* o Grassfire.

En todo caso, y ya que no contamos con robots físicos para el TFM, estas son simplemente algunas ideas importantes sobre la navegación que podrían implementarse si se decidiera ampliar el trabajo.

También tengo que añadir que, a pesar de que no realicemos un trabajo de navegación con robot físicos, el desarrollo del mapa topológico ha estado siempre enfocado en mejorar el entendimiento de un robot sobre el terreno, influyendo la navegabilidad en la selección de los nodos definidos anteriormente en 2.3.2 y de los landmarks escogidos para representar cada nodo del mapa topológico.

# Capítulo 3

## Generación del dataset de imágenes de landmarks para el entrenamiento y evaluación de los reconocedores

### 3.1. Definición del dataset

Para empezar a entrenar la red neuronal es necesario tener un dataset rico en imágenes tomadas desde distintos ángulos y con distinta iluminación. No obstante, el primer paso es organizar y confeccionar el nivel jerárquico de carpetas que la red neuronal leerá. De esta manera, dividí el dataset en “train”, “validation” y “test” como puede verse en la Figura 3.1.

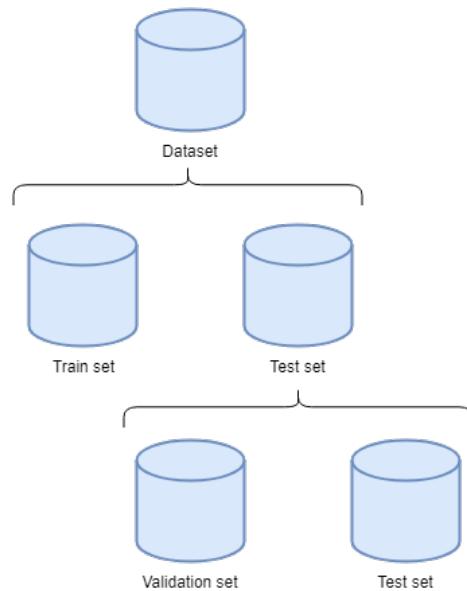


Figura 3.1: Dataset para Redes Neuronales

Esta organización mostrada en la Figura 3.1 es la organización habitual para

entrenar una red neuronal mediante *holdout validation*. Además, para entrenar un clasificador multiclasa mediante el software que presentaremos en Capítulo 4 hay que realizar una jerarquía de carpetas en las que se cree una carpeta para cada clase/etiqueta que queramos entrenar para clasificar. De esta manera he creado una división en carpetas en la que cada nodo del mapa topológico se corresponde a una clase del clasificador de la siguiente manera:

- *dataset*: el dataset en su totalidad.
  - *train*: las imágenes de entrenamiento de la red neuronal.
    - *Nodo1*: las imágenes de entrenamiento del nodo 1.
    - ...
    - *Nodo7*: las imágenes de entrenamiento del nodo 7.
  - *validation*: las imágenes de validación de la red neuronal.
    - *Nodo1*: las imágenes de validación del nodo 1.
    - ...
    - *Nodo7*: las imágenes de validación del nodo 7.
  - *test*: los vídeos de test finales.
    - *originals*: los vídeos originales de test sobre los que se les pasará frame a frame la red neuronal para comprobar su rendimiento.
    - *results*: los vídeos generados por los scripts en los que se clasifica frame a frame un vídeo de la carpeta “*originals*”.

## 3.2. Toma de imágenes

El proceso de toma de imágenes es muy simple; he grabado una serie de vídeos y he guardado cada uno de los frames de esos vídeos como una imagen.

Para realizar esto me he apoyado en un Jupyter Notebook implementado en python por mi llamado “*Dataset Refinement.ipynb*”.

Para usar el cuaderno jupyter lo primero que hay que hacer es renombrar los vídeos de los que queremos extraer imágenes. Para ello debemos llamarlos igual que el nodo al que pertenece el vídeo junto a una barra baja y a un número para poner un orden en el que se irá accediendo a estos vídeos. De esta manera tendremos vídeos con nombre “Nodo{Número del nodo} \_ {Otro número para indicar el orden de acceso}”.

El cuaderno jupyter va leyendo vídeo a vídeo los frames y los va guardando automáticamente en la carpeta correspondiente a su nodo, no sin antes realizar las operaciones de conversión que serán explicadas en el siguiente apartado 3.3.

### 3.3. Tratamiento de las imágenes

La cámara con la que se realizaron los vídeos generó imágenes de tamaño 1920x1080. No podemos pasárselas a la red neuronal imágenes tan grandes para que entrene con ellas pues el tiempo que tardaría en entrenar podría ser de meses.

He reducido el tamaño de las imágenes y he probado con distintos formatos, siempre manteniendo la imagen con un tamaño rectangular tal y como se tomaron con la cámara y siempre intentando mantener los objetos de la imagen reconocibles.

Se probaron distintos formatos: 960 x 540, 480 x 270 y 240 x 135, pero seguía siendo demasiado grande y la red tardaba aún bastante tiempo en realizar el entrenamiento, por lo que se optó por disminuir el tamaño aún más hasta los 128 x 72, en los que la red solo tardaba unas horas en realizar el entrenamiento con el dataset mientras se seguían manteniendo los bordes que delimitaban los objetos aún visibles.

### 3.4. Contenido del Dataset

El primer dataset con el que empecé a entrenar la red neuronal contaba con 500 imágenes para entrenamiento y 50 para validación por nodo y, a medida que la red neuronal iba mostrándome sus resultados, había que añadir al dataset más imágenes de algunos escenarios que no era capaz de generalizar.

El dataset final de este proyecto está compuesto por 46.772 imágenes de las que 42.093 han sido seleccionadas para entrenamiento y 4.679 para validación. Esto supone que el 90 % de las imágenes totales del dataset han sido usadas para entrenamiento y el 10 % de las imágenes totales del dataset han sido usadas para la validación. Dicha división ha sido representada en la Figura 3.2.

#### 3.4.1. Training

Ya que tenemos alrededor de 42.000 imágenes y 7 nodos, para el entrenamiento idílicamente a cada nodo le tocárían, por media, unas 6.000 imágenes. No obstante, realizando iteraciones para mejorar los resultados de la red neuronal me he visto obligado a tomar más fotos de algunos nodos que no generalizaban bien. En cambio, otros nodos generalizaban bien desde un principio y no necesitaban tantas imágenes para llegar a buenos resultados.

El resultado final ha sido el siguiente:

- *Nodo 1*: 4761 imágenes
- *Nodo 2*: 5901 imágenes
- *Nodo 3*: 4140 imágenes
- *Nodo 4*: 5391 imágenes



Figura 3.2: División del dataset para este proyecto

- *Nodo 5*: 9701 imágenes
- *Nodo 6*: 7342 imágenes
- *Nodo 7*: 4857 imágenes

Los nodos han necesitado un mínimo de 4.000 imágenes para generalizar y converger en buenos resultados mientras que el nodo 5 ha necesitado muchas más imágenes para poder generalizar. En el Capítulo 4 hablaremos en más detalle del proceso de entrenamiento y de las dificultades en la generalización.

### 3.4.2. Validation

En lo que respecta a la validación, simplemente he tomado un 10 % de imágenes aleatorias de cada nodo del dataset final. El resultado es el siguiente:

- *Nodo 1*: 529 imágenes
- *Nodo 2*: 656 imágenes
- *Nodo 3*: 460 imágenes
- *Nodo 4*: 600 imágenes
- *Nodo 5*: 1078 imágenes
- *Nodo 6*: 816 imágenes
- *Nodo 7*: 540 imágenes

# Capítulo 4

## Clasificador: Deep Learning + CNNs

### 4.1. Implementación

A continuación detallaré todos los aspectos importantes de la implementación del clasificador que he construido mediante las técnicas de Deep Learning. Detallaré los lenguajes y frameworks con los que he construido la red neuronal, asimismo como su arquitectura o los resultados en el entrenamiento.

La implementación puede encontrarse en el Jupyter Notebook adjuntado a esta memoria llamado “Neural Network Training - Desktop.ipynb” o “Neural Network Training - Google Colab.ipynb” si desea ser ejecutado en Google Colab, siendo este último con el que hemos realizado en este proyecto.

#### 4.1.1. Herramientas

Para implementar la red neuronal he decidido usar TensorFlow apoyándome en Keras.

##### ■ TensorFlow

TensorFlow es una librería de código abierto desarrollada por Google. Esta librería permite construir redes neuronales y entrenarlas tanto en CPUs como en GPUs.

Parto de una ligera experiencia en TensorFlow; en la asignatura de Visión por Computador de este mismo máster se realizó una pequeña red neuronal para resolver el problema de CIFAR-100.

##### ■ Keras

Keras es una API de alto nivel que facilita la implementación de redes neuronales escritas en TensorFlow (entre otros).

Tanto en Keras como en TensorFlow existe una gran comunidad de desarrolladores que comparten todo su conocimiento en redes neuronales en forma de foros,

guías, tutoriales, blogs y competiciones. Toda esta documentación facilitará la implementación, ya que habrá muchos recursos que explicarán en detalles las capas, técnicas y complementos usados para realizar una red neuronal más eficiente.

#### 4.1.2. Sistema

Esta red neuronal será ejecutada y entrenada en Google Colab.

Google Colab permite ejecutar Jupyter Notebooks en la nube. Su existencia se debe a la apuesta por parte de Google en Machine Learning y, en especial, en promover el uso de TensorFlow y Keras entre los desarrolladores Deep Learning.

El entorno es completamente gratuito, ya cuenta con TensorFlow y Keras instalado y da acceso a una Tesla K80 GPU, lo que se traduce en alrededor de 15 GB de tarjeta gráfica gratuita para entrenar nuestra red.

De esta manera, almacenaremos nuestro dataset en Google Drive y desde Google Colab accederemos a dicho dataset. Tristemente, el acceso de Google Colab al dataset de Google Drive llevará tiempo, y la tarjeta gráfica tardará en procesar las imágenes pudiendo tardar de 3 a 7 horas en realizar el primer epoch de entrenamiento (El resto de epoch irían más rápido ya que las imágenes ya estarían en memoria). No obstante, este enfoque de usar la nube con una tarjeta gráfica potente para entrenar nuestra red es mucho mejor que el uso de una CPU de un ordenador personal como era la alternativa.

#### 4.1.3. Arquitectura de la red

El diseño final de la arquitectura de la red neuronal y con la que mejores resultados he obtenido puede observarse en la Tabla 4.1.

Se trata de una red neuronal secuencial con seis capas convolucionales seguidas todas ellas de una normalización batch, activación ELU y capa de max pooling. Para conseguir un clasificador multiclase al final de la red nos encontramos con una capa densa de 7 neuronas que se corresponden a las 7 clases que puede discernir esta red.

A medida que vamos avanzando en profundidad por la red vamos disminuyendo el número de neuronas. De esta manera empezamos en 128 neuronas con una capa convolucional, le sigue una de 64, luego varias de 32 y la capa densa final de 7.

Como optimizador para esta red hemos usado Adam [1], que en la actualidad es el mejor método para que toda red converja al óptimo global.

### 4.2. Técnicas usadas

A continuación explicaré algunas técnicas usadas en la elaboración de esta red neuronal que son merecedoras de ser explicadas pues intervienen directamente en el proceso de generalización de la red y contribuyen a un incremento en la tasa de aciertos en los test finales.

Tabla 4.1: Arquitectura de la red neuronal

Layer	Neuronas	Configuración	Output Shape	Parámetros
Conv2d	128	Región = (20,20)	(None, 72, 128, 128)	153728
Batch Normalization			(None, 72, 128, 128)	512
Activación		ELU	(None, 72, 128, 128)	0
Max Pooling 2D		(2,2)	(None, 36, 64, 128)	0
Conv2d	64	Región = (10,5)	(None, 36, 64, 64)	409664
Batch Normalization			(None, 36, 64, 64)	256
Activación		ELU	(None, 36, 64, 64)	0
Max Pooling 2D		(2,2)	(None, 18, 32, 64)	0
Conv2d	32	Región = (10,5)	(None, 18, 32, 32)	102432
Batch Normalization			(None, 18, 32, 32)	128
Activación		ELU	(None, 18, 32, 32)	0
Max Pooling 2D		(2,2)	(None, 9, 16, 32)	0
Conv2d	32	Región = (5,5)	(None, 9, 16, 32)	25632
Batch Normalization			(None, 9, 16, 32)	128
Activación		ELU	(None, 9, 16, 32)	0
Max Pooling 2D		(2,2)	(None, 4, 8, 32)	0
Conv2d	32	Región = (3,3)	(None, 4, 8, 32)	9248
Batch Normalization			(None, 4, 8, 32)	128
Activación		ELU	(None, 4, 8, 32)	0
Max Pooling 2D		(2,2)	(None, 2, 4, 32)	0
Conv2d	32	Región = (3,3)	(None, 2, 4, 32)	9248
Batch Normalization			(None, 2, 4, 32)	128
Activación		ELU	(None, 2, 4, 32)	0
Flatten			(None, 256)	0
Densa	7		(None, 7)	1799

### 4.2.1. CNNs - Convolutional Neural Networks

Las capas convolucionales empezaron a ser usadas en redes neuronales tras el éxito de AlexNet [6], donde se comenzaron a usar con buenos resultados.

Normalmente en una capa de una red neuronal las neuronas se encuentran todas colocadas en fila. Mediante las capas convolucionales pasamos de tener un array de pesos a una matriz, donde las neuronas están ordenadas de forma espacial en un plano de dos dimensiones. Aprovechando esta gestión espacial, las neuronas observan una región o kernel de la imagen, encontrando patrones comunes entre todas las imágenes que les sirven de training y de esta manera generalizando las imágenes. Este funcionamiento puede verse representado en la Figura 4.1.

Cuanto mayor es la región más tarda la red neuronal en asignar cada peso a las neuronas, es por esto por lo que se trabaja con dimensiones de región pequeñas nunca mayores de (20,20). Además, en [7] se demuestra que una red neuronal con capas convolucionales de región (3,3) podrían resolver cualquier problema con

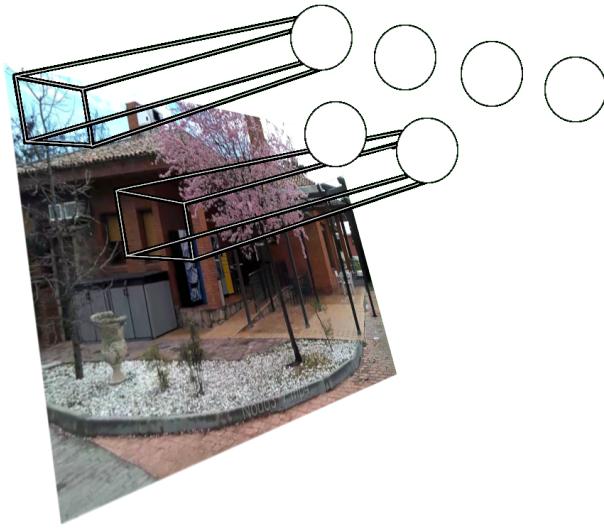


Figura 4.1: Funcionamiento de las capas convolucionales

la profundidad de red suficiente (siguiendo su arquitectura de Microsoft ResNet). Inspirado en este paper he configurado las tres últimas capas de la red neuronal con una región de (5,5) y (3,3).

Con el uso de las capas convolucionales conseguimos un menor número de parámetros, pues las capas densas suponen un gasto innecesario en parámetros, lo que conlleva un tiempo de entrenamiento mayor y un rendimiento peor.

Gracias a estas capas convolucionales tan solo tenemos 712.519 parámetros, cuando con capas densas llegaríamos a tener millones.

#### 4.2.2. Data augmentation

Muchas veces los datos con los que contamos para el entrenamiento no son suficientes para entrenar una red neuronal y, en el supuesto caso de que tuviésemos suficientes datos, tener más datos siempre mejora los resultados de este tipo de clasificadores.

Gracias al data augmentation podemos “inventarnos” datos nuevos a partir de otros datos originales de nuestro dataset. En el caso de que los datos sean imágenes lo que se puede hacer es generar nuevas imágenes realizando transformaciones a imágenes originales de nuestro dataset jugando con los contrastes, girándolas, doblándolas, invirtiéndolas, etc. De esta manera se comienza trabajando con un dataset de mayor tamaño y se consiguen mejores resultados.

No obstante, aunque data augmentation sea muy configurable pudiendo indicarle el tipo de transformaciones que deseamos realizar a las imágenes, el diseñador de la red ha de escoger las operaciones que si contribuyen a la mejora de la red, descartando transformaciones que no aporten nada o incluso que pudieran dañar el resultado

final.

Como ejemplo se puede observar la Figura 4.2 en la que podemos ver como mediante data augmentation se ha invertido horizontalmente (imagen espejo) una imagen de nuestro dataset. Esta transformación no contribuye en nada a nuestro proyecto; en un proyecto en el que se clasificasen objetos se podría usar este enfoque ya que un objeto sigue siendo el mismo objeto incluso invertido, pero en este caso si invertimos un landmark, como es el parking de la Figura 4.2, estamos destruyendo el propio landmark ya que habríamos perdido la ubicación relativa en la que podrían estar los objetos de la escena, posiciones que la red podría generalizar para identificarla.



Figura 4.2: Generación de una nueva imagen invertida (imagen espejo, “flip”) a partir de otra imagen.

Tras un análisis de las posibles opciones que podrían usarse para mejorar nuestro dataset he llegado a la conclusión de que las funciones que más me interesan para la red son los giros. Los giros no alteran las posiciones de los objetos de las escenas ni afectan a los colores o estructuras de la imagen. En la Figura 4.3 se puede observar como sería el proceso de data augmentation que hemos realizado para este proyecto.

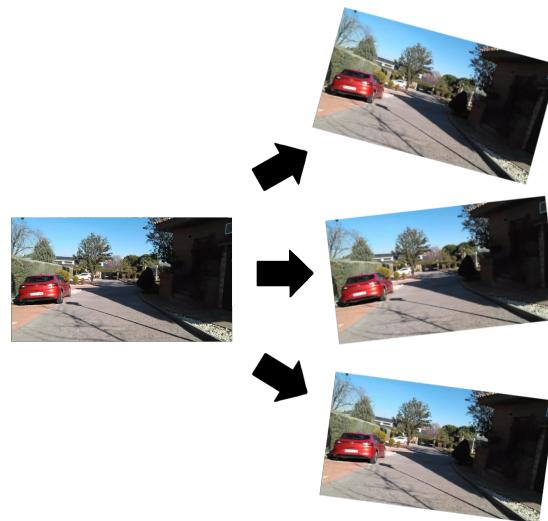


Figura 4.3: Generación de nuevas imágenes con giros a partir de una imagen original.

Mediante este proceso ganamos imágenes para nuestro dataset con una perspectiva distinta a la original, por lo que la red generalizará mejor. En el cuaderno de jupyter en el que hemos entrenado la red podemos ver que hemos seleccionado giros de como máximo  $180^\circ$ , teniendo así un enfoque completo de la escena, incluso con imágenes invertidas verticalmente.

### 4.2.3. Max pooling

Existe una capa de “Max Pooling” en keras que permite disminuir las dimensiones de la imagen a medida que va pasando por las distintas capas de nuestra red.

Haciendo más pequeña las imágenes las neuronas generalizan los cambios de escala de las imágenes, por lo que se obtienen mejores resultados ya que los cambios de dimensiones de las estructuras se generalizan.

Además, trabajar con imágenes más pequeñas disminuye el número de parámetros de la red, por lo que el entrenamiento será ejecutado más rápidamente.



Figura 4.4: Ejemplo del funcionamiento de las capas Max Pooling 2D

En la Figura 4.4 puede observarse como funciona este proceso. En la red que hemos entrenado he usado un max pooling que disminuye a la mitad el tamaño de la imagen. Esto puede verse en la Tabla 4.1 donde la columna *Configuración* siempre se encuentra en las capas de Max Pooling con un valor de (2,2) y en la columna *Output Shape* donde tras un Max Pooling disminuyen las dimensiones del output entre capas (de, por ejemplo, (*None*, 72, 128) a (*None*, 36, 64)).

### 4.2.4. Dropout

En las primeras arquitecturas que implementé usé capas de Dropout [4].

Algunas neuronas tienden a especializarse demasiado y a ser de vital importancia para el funcionamiento de la red adquiriendo un peso muy elevado mientras que otras neuronas de su misma capa tienden a volverse innecesarias con un peso diminuto. Esto es muy malo para una red neuronal, tanto en fase de entrenamiento como en fase test, pues los resultados serían mucho mejores si todas las neuronas trabajaran en conjunto.

Para solventar este problema se inventó el dropout. Este permite apagar un tanto por ciento de neuronas durante la fase de entrenamiento, obligando al resto de neuronas a trabajar más para poder clasificar los datos de entrada.

No obstante, en cierta literatura como [5] se desaconseja el uso de Dropout junto a capas convolucionales debido, en parte, a la gestión espacial de las neuronas convolucionales. Debido a esto, a medida que la arquitectura de la red iba mejorando fui poco a poco probando a quitar las capas de dropout y sustituyéndolas por capas de normalización batch. Como resultado, y a medida que iba experimentando con nuevas arquitecturas, vi que la tasa de aciertos convergía mejor con batch normalization y vi la innecesidad de usar dropout, llegando al punto de su eliminación en la arquitectura final sustituyéndolos por normalización batch.

#### 4.2.5. Batch Normalization

La nomalización batch [13] es una técnica y capa de red neuronal muy utilizada que mejora el rendimiento de las redes neuronales disminuyendo el tiempo de entrenamiento normalizando los output de cada capa.

Estas capas cogen los output de las neuronas y los normalizan antes de entrar en la función de activación (por eso se encuentra en medio de dichas capas en nuestra arquitectura). Como resultado, se pierden los valores extremos o atípicos de las salidas de las neuronas, lo que se traduce en un entrenamiento más rápido, pues los valores de activación de la neurona no oscilan tanto cuando son asignados a cada neurona en el entrenamiento, y un mejor rendimiento de la red.

#### 4.2.6. Activaciones ELU

Para las funcions de activación de las neuronas he usado ELUs [12].

Estas funciones de activación dan mejores resultados que las RELU, a costa de más tiempo de computo.

### 4.3. Entrenamiento

En la Tabla 4.2 puede verse parte del entrenamiento de la red neuronal. El entrenamiento al completo de la red puede verse en el Apéndice A.

La primera epoch siempre tarda alrededor de tres horas ya que es la epoch en la que las imágenes son leídas por la GPU y a partir de ahí la potencia de la concurrencia de las unidades gráficas permiten un entrenamiento de cinco minutos por epoch. Google Colab reinicia cada veinticuatro horas sus GPUs, por lo que cada día he tenido que esperar una ventana de tres horas para volver a meter las imágenes en la GPU y continuar el entrenamiento desde donde lo dejaba el anterior día.

Esta red ha sido entrenada durante un periodo de diecisiete horas (64026 segundos). En esas diecisiete horas ha sido entrenada en 90 epochs, con la que se ha obtenido una accuracy del 99,23 %.

El proceso de entrenamiento de una red neuronal es simple:

1. Primero, el dataset es guardado en batches. Para nuestro entrenamiento he seleccionado batches de tamaño 124. Por así decirlo, he guardado las imágenes en “bolsas” de 124 imágenes para pasárselas todas juntas a la red.
2. A continuación, se ejecuta la red con los pesos actuales de la red con uno de los batches. Cuando se finaliza el batch se comprueba la accuracy que se ha obtenido y se retroalimentan los pesos de la red con esa información para mejorar en futuras predicciones. Se continua hasta que no haya más batches.

Esto justifica el uso de los batches ya que de no usarlos se realizaría la retroalimentación con cada imagen, lo que sin duda alargaría en el tiempo el entrenamiento. Al usar batches se va acumulando la información de la tasa

Tabla 4.2: Entrenamiento de la red

Epoch	Loss	Accuracy	Val-Loss	Val-Accuracy	Tiempo (segundos)
1	1,1183	0,5995	1,0262	0,6968	14260
2	0,9144	0,6698	3,4629	0,3520	221
3	0,4787	0,8390	1,5481	0,6467	219
4	0,3493	0,8906	1,1629	0,6619	217
5	0,2942	0,9106	0,7883	0,7449	216
6	0,2347	0,9331	0,8077	0,7824	214
7	0,2056	0,9434	0,2193	0,9335	216
8	0,2030	0,9462	0,4999	0,8498	215
9	0,1772	0,9560	0,4610	0,8617	215
10	0,1678	0,9593	0,2026	0,9409	214
11	0,1667	0,9590	0,9479	0,7557	215
12	0,1497	0,9667	0,1069	0,9829	215
13	0,1501	0,9658	0,4005	0,9161	221
14	0,1546	0,9668	0,1360	0,9730	216
15	0,1487	0,9689	0,1152	0,9813	217
16	0,1416	0,9721	0,2202	0,9416	217
17	0,1296	0,9759	0,1740	0,9627	215
18	0,1516	0,9680	0,5211	0,8505	215
19	0,1285	0,9778	0,1822	0,9550	217
...					
90	0,0654	0,9943	0,0680	0,9923	204

de ciertos y se pasa todo junto para la retroalimentación, disminuyendo así el tiempo de entrenamiento y mejorando el rendimiento de la red.

3. Una vez se ha entrenado la red con todos los batches del dataset se hacen las pruebas de validación. Una vez en cada epoch al finalizar.

Una vez se han realizado las pruebas de validación tendremos cuatro valores:

- **Accuracy:** la tasa de aciertos durante entrenamiento, esto es, la tasa de aciertos al haber ido pasando cada batch o, si se entiende mejor, la tasa de aciertos usando los datos de training.
- **Val.Accuracy:** la tasa de aciertos usando los datos de validación.
- **Loss:** El valor Loss usando los datos de training.
- **Val.Loss:** El valor Loss usando los datos de validación.

El valor Loss un valor escalar que indica cuan cercanas son las predicciones del valor real de las clases. Siempre se intenta minimizar este valor. También es llamado *Mean Squared Error* (MSE) o, en Keras, *Categorical Cross Entropy*.

Con esta información hemos realizado las gráficas que pueden verse en las Figuras 4.5, 4.6 y 4.7 donde se muestra la evolución en el tiempo de estos cuatro valores a medida que avanzamos en el entrenamiento.

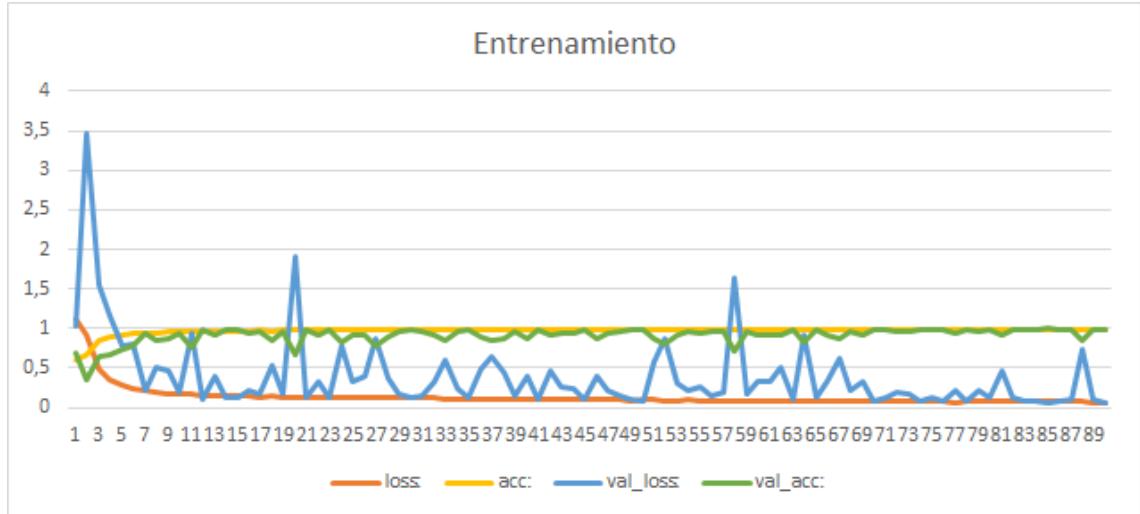


Figura 4.5: Evolución del entrenamiento en cada epoch mostrando el *Loss* y la *Accuracy*

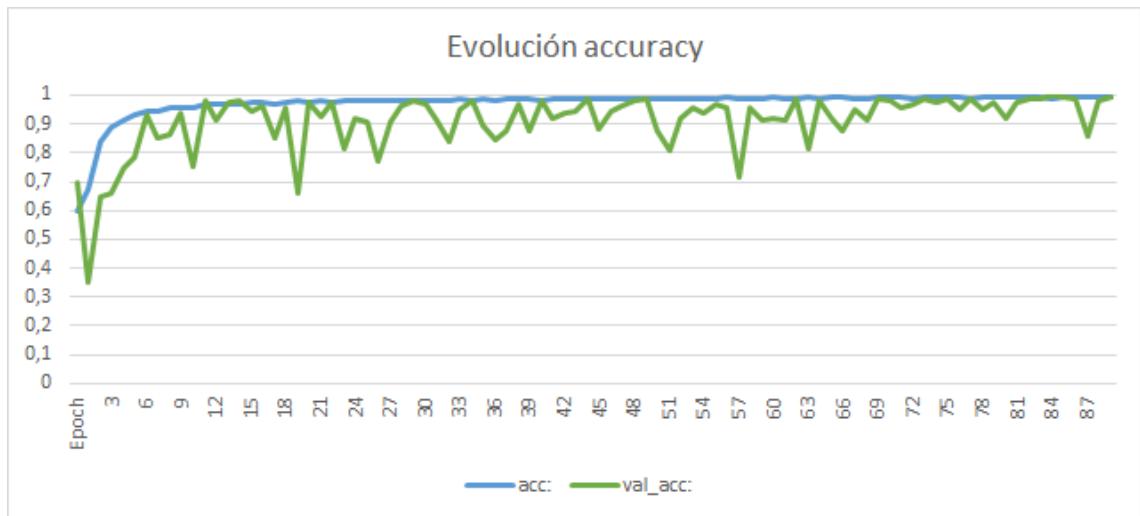


Figura 4.6: Evolución de la *Accuracy* en cada epoch

En estas gráficas se puede observar que no tenemos ni un exagerado underfitting (generaliza demasiado) ni un gran overfitting (se ajusta demasiado al dataset de training). Esto puede verse en la Figura 4.6 donde se ve que los valores de la accuracy en validación son siempre valores cercanos al accuracy en training (excepto contadas ocasiones completamente despreciables). Que los valores de *val\_acc* oscilen por arriba y por abajo de los valores *acc* indica la buena salud de nuestra red y que esta ha generalizado correctamente.

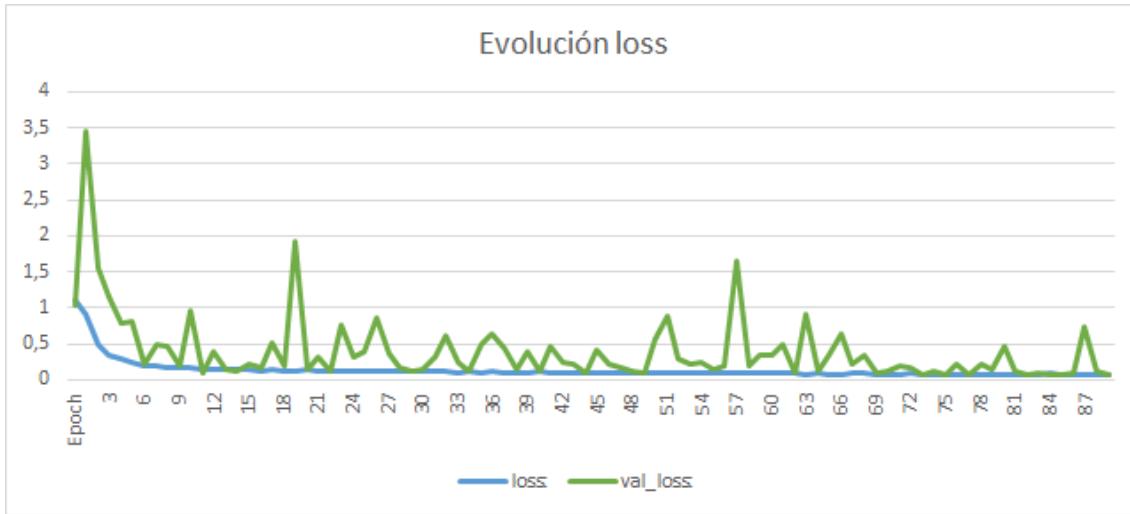


Figura 4.7: Evolución del valor *Loss* en cada epoch

El resultado final del entrenamiento ha sido la última red neuronal mostrada en la Tabla 4.2: tras el epoch 90 tenemos una red que adquiere en entrenamiento un valor Loss de 0,0654 y un valor en tasa de aciertos del 0,9943. En sus pruebas de validación esta misma red dió un valor de Loss de 0,0680 y una tasa de aciertos del 0,9923, ambos valores muy cercanos a los valores vistos en el entrenamiento durante esa misma epoch.

# Capítulo 5

## Validation: test estáticos

### 5.1. Jupyter notebook

Para realizar las pruebas de validación, a parte de las funciones de las que provee Keras en el entrenamiento para validar la red a medida que es entrenada en cada epoch, hemos realizado una serie de scripts que pueden verse en el Jupyter Notebook llamado “*Neural Network Validation - Static Test.ipynb*”.

### 5.2. Conjunto de datos de validación

He usado *holdout validation* para comprobar la eficiencia de la red neuronal entrenada.

En el apartado 3.4 expliqué como se realizó la división de las imágenes del dataset en 90 % para entrenamiento y un 10 % para validación. Como fue explicado en el apartado 3.4.2 en total tenemos el siguiente número de imágenes para validación:

- *Nodo 1*: 529 imágenes
- *Nodo 2*: 656 imágenes
- *Nodo 3*: 460 imágenes
- *Nodo 4*: 600 imágenes
- *Nodo 5*: 1078 imágenes
- *Nodo 6*: 816 imágenes
- *Nodo 7*: 540 imágenes

### 5.3. Proceso de validación

Keras realiza el proceso de validación mediante *holdout validation* al finalizar cada epoch en el entrenamiento para comprobar la eficiencia de la red. Esta función automática de Keras puede verse en la Figura 5.1.

```
Epoch 1/10
339/339 [=====] - 216s 639ms/step - loss: 0.0756 - acc: 0.9919 - val_loss: 0.4593 - val_acc: 0.9207
Epoch 2/10
339/339 [=====] - 210s 620ms/step - loss: 0.0767 - acc: 0.9917 - val_loss: 0.1276 - val_acc: 0.9769
Epoch 3/10
339/339 [=====] - 207s 611ms/step - loss: 0.0752 - acc: 0.9916 - val_loss: 0.0772 - val_acc: 0.9901
Epoch 4/10
339/339 [=====] - 209s 615ms/step - loss: 0.0698 - acc: 0.9937 - val_loss: 0.0879 - val_acc: 0.9849
Epoch 5/10
339/339 [=====] - 209s 617ms/step - loss: 0.0868 - acc: 0.9875 - val_loss: 0.0599 - val_acc: 0.9982
Epoch 6/10
339/339 [=====] - 208s 613ms/step - loss: 0.0695 - acc: 0.9942 - val_loss: 0.0706 - val_acc: 0.9932
Epoch 7/10
339/339 [=====] - 207s 611ms/step - loss: 0.0727 - acc: 0.9922 - val_loss: 0.0950 - val_acc: 0.9866
Epoch 8/10
339/339 [=====] - 207s 611ms/step - loss: 0.0765 - acc: 0.9910 - val_loss: 0.7263 - val_acc: 0.8562
Epoch 9/10
339/339 [=====] - 209s 617ms/step - loss: 0.0690 - acc: 0.9937 - val_loss: 0.1127 - val_acc: 0.9794
Epoch 10/10
339/339 [=====] - 204s 603ms/step - loss: 0.0654 - acc: 0.9943 - val_loss: 0.0680 - val_acc: 0.9923
```

Figura 5.1: Output en el entrenamiento de una red neuronal con Keras

Keras proporciona, además, un método llamado *.evaluate\_generator* que tiene como parámetros las imágenes con las que deseamos validar nuestra red neuronal (en forma de *ImageDataGenerator* y con el tamaño del *batch* entre otras configuraciones) y al cabo de pocos minutos nos devuelve los resultados en *loss* y *accuracy*. Los resultados del método *.evaluate\_generator* pueden verse en la Figura 5.2.

```
Found 4679 images belonging to 7 classes.

Validation starts
Validation finishes

Validation with model.evaluate_generator() took 305.21905612945557 seconds

Accuracy: 0.9908100093218678 | Loss: 0.07248106863183276
```

Figura 5.2: Output del método *.evaluate\_generator*

Este método tarda muy poco en ejecutarse, pero no imprime más información que los resultados en forma de *loss* y *accuracy*. Debido a esto he escrito un pequeño script que imprime por pantalla más información, como la matriz de confusión, a costa de más tiempo de ejecución.

### 5.4. Resultados

La red neuronal que hemos entrenado ha conseguido un accuracy del 99 % como puede verse en el output de los scripts de nuestro Jupyter Notebook en la Figura 5.3 o en el entrenamiento mostrado en la anterior Figura 5.1.

```
Total Images: 4679
True positives: 4636
Accuracy: 0.9908100021372088
Accuracy: 0.9908100093218678 | Loss: 0.07248106863183276
```

Figura 5.3: Output del Jupyter Notebook con los resultados de nuestra red neuronal mediante los métodos de kera y los scripts implementados

Estos scripts también generan dos matrices de confusión mostradas en la Figura 5.4 y la Figura 5.5, correspondiéndose esta última a la matriz de confusión normalizada que muestra el porcentaje de imágenes que han sido clasificadas correcta e incorrectamente dando una visión individual de la tasa de aciertos por cada nodo.

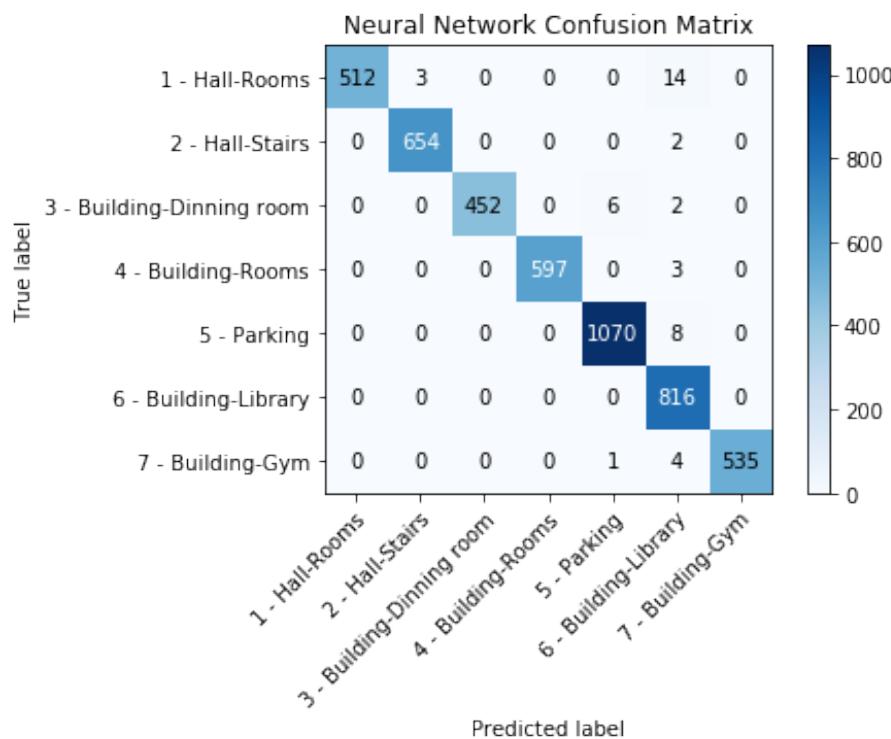


Figura 5.4: Matriz de confusión

En la matriz de confusión normalizada podemos ver que el nodo con menor tasa de aciertos es el nodo 1, las habitaciones del hall, con un 97% de accuracy. Que el nodo con menor accuracy de una tasa de aciertos tan alta es una buena señal del correcto funcionamiento de la red en pruebas test.

Podemos ver también que la mayoría de los falsos positivos se dan en el nodo 6, la biblioteca. Esto es debido a que es el nodo más difícil de clasificar pues comparte estructuras con el resto de nodos. Si se tomanen más imágenes de dicho nodo y se entrenase a la red con ellas tal vez se podría disminuir el número de falsos positivos. No obstante, con un total de 33 imágenes mal clasificadas en ese nodo del resto de 3863 (un 0.008 % de las imágenes), podemos despreciar este error.

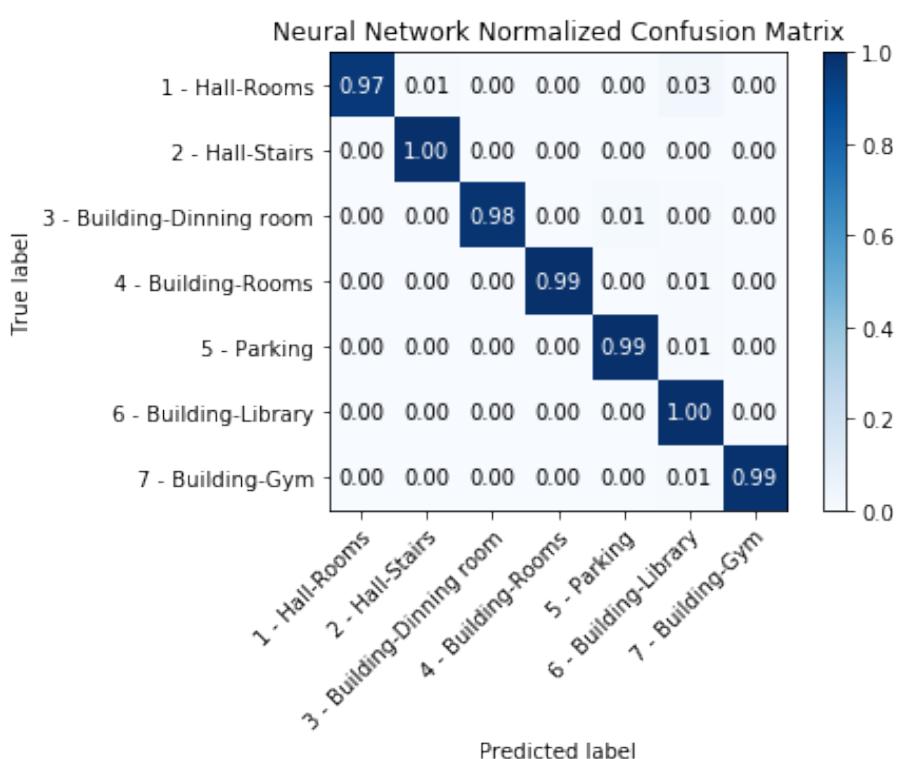


Figura 5.5: Matriz de confusión normalizada

# Capítulo 6

## Test: tests dinámicos

### 6.1. Jupyter notebook

Para realizar las pruebas de test Keras no provee de ninguna facilidad, a excepción del método que permite a la red neuronal predecir la etiqueta de las imágenes mostrada.

He implementando varios scripts en python que pueden verse en el Jupyter Notebook llamado “*Neural Network Test - Dynamic Test.ipynb*” que permiten realizar estas pruebas, pasando la red neuronal frame a frame por un vídeo.

### 6.2. Conjunto de datos de test

Contamos con un vídeo de tres minutos en el que se recorre el mapa topológico mientras la cámara apunta a landmarks visuales reconocibles por la red neuronal.

Estos 3:03 minutos de vídeo a 30 frames por segundo suponen un total de 5511 frames que la red neuronal tendrá que analizar.

### 6.3. Proceso de test

Keras no proporciona ninguna ayuda para realizar las pruebas de tipo test, por lo que he escrito mis propios scripts en python para realizar estas pruebas en vídeos. Puede verse en el Algoritmo 1 el proceso base de como hago estas pruebas:

Este proceso por el que se realizan las pruebas explicado textualmente es el siguiente:

1. Seleccionamos un vídeo y leemos cada uno de sus frames.
2. La red neuronal recibirá estos frames como inputs y devolverá las probabilidades de que estos frames pertenezcan a cada una de las clases.

**Algoritmo 1** *dinamic\_test(name\_video)*


---

```

create_temporal_folder()
video = read_video(name_video)
frame = video.next()
while (frame) do
    label = predict_label(frame)
    new_frame = write_label_in_frame(label, frame)
    save_in_temporal_folder(new_frame)
    frame = video.next()
end while
create_new_video()
delete_temporal_folder()

```

---

3. La clase con la mayor probabilidad será considerada la etiqueta correcta del frame. Esta clase predicha será comparada con la clase real"(la etiqueta que debería ser) para calcular la accuracy en tiempo de test.
4. Guardamos en una carpeta temporal el frame con la etiqueta a la que pertenece escrita en la imagen.
5. Seguimos iterando hasta que no queden frames que tratar.
6. Generamos y guardamos un nuevo vídeo con todos los frames tratados con su etiqueta de la carpeta temporal.
7. Borramos la carpeta temporal y devolvemos la accuracy de la operación.

El nuevo vídeo muestra la evolución en el tiempo de las predicciones de la red neuronal y sirve para comprobar la generalización en un entorno dinámico.

Gracias a estas pruebas, en un supuesto caso de que implementásemos este sistema en un robot real, podemos ver como el robot se relacionaría con el entorno, los errores de clasificación que tendría y las dificultades a las que se debería enfrentar para encontrarse en el entorno.

No obstante, a pesar de las ventajas de realizar estas pruebas, este enfoque tiene dos problemas que han de ser mencionados:

1. **Tiempo de ejecución:** Estas pruebas tardan mucho tiempo en ejecutarse; estamos hablando de realizar cinco operaciones muy costosas en cada uno de los fotogramas: leer el frame de 1920x1080, redimensionar el frame de 1920x1080 a una imagen de 128x72 para que la red neuronal pueda leerlos, redimensionar el frame de 1920x1080 para que no ocupe demasiado espacio en el disco, escribir sobre imágenes y guardar nuevas imágenes en disco. Todas estas operaciones para cada uno de los frames.

Experimentalmente hemos comprobado que en un segundo se pueden tratar uno o dos frames. Esto quiere decir que para tratar un vídeo de 5000 frames se tardan alrededor de 45 minutos o 1 hora.

2. **Mapeo:** Es necesario mapear todos los frames del vídeo indicando la etiqueta real del frame para poder hacer una medición de la accuracy con las etiquetas predichas por la red neuronal.

Para hacer esto hay que realizar una ejecución preliminar de estos mismos scripts y generar un vídeo que indique el número del frame que se encuentra mostrando en todo momento. Mediante este vídeo podremos estudiar los frames y los cambios de un nodo a otro para luego hacer un mapeo para generar, ya esta vez, el vídeo correcto.

## 6.4. Resultados

En la Figura 6.1 puede verse el output de los scripts implementados, mostrando que nuestra red neuronal ha alcanzado una tasa de aciertos del 90 % al analizar el vídeo grabado.

```
Working on frame nº5511
New Video - Working on frame nº5511
Video "test2.mp4" ready
-True Positives: 5005
-Total Number Frames: 5512
-Accuracy: 0.9080188679245284
```

Figura 6.1: Output de las pruebas test

Este 10 % de errores supone que la red neuronal erra unos 540 frames (507 frames, si somos precisos), lo que suponen unos 18 segundos de vídeo a 30 frames/segundo. En el vídeo generado con el análisis de la red neuronal se puede observar que la mayoría de estos segundos se tratan de momentos en los que se producen superposición de landmarks visuales como puede verse en la Figura 6.2, donde las predicciones de la red neuronal oscilan entre varios nodos, pues no sabe decidirse en cual de los nodos clasificar una imagen.

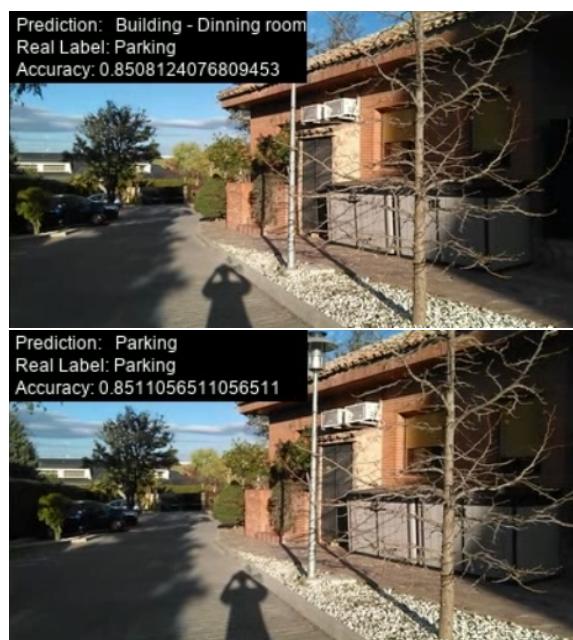


Figura 6.2: Superposición de nodos entre el nodo 3 (Building - Dinning room) y el nodo 5 (Parking)

# Capítulo 7

## Conclusiones y líneas futuras de trabajo

### 7.1. Conclusiones

En este proyecto he mostrado la potencia del deep learning al construir de manera simple una red neuronal para clasificar e identificar los nodos de un mapa topológico de landmark visuales.

Gracias a la red neuronal que hemos entrenado hemos llegado a unos resultados bastante prometedores para un clasificador de landmarks visuales.

Tomando como premisa que un ser humano no confundiera nunca los landmark visuales podríamos considerar que la tasa de aciertos o accuracy de nuestro sistema sería el siguiente:

$$A_{humano} \approx 100\%, A_{train} \approx 99\%, A_{test} \approx 90\%$$

Mirando los datos de tasa de errores:

$$E_{humano} \approx 0\%, E_{train} \approx 1\%, E_{test} \approx 10\%$$

Estos datos nos indican que hemos llegado al límite de tasa de aciertos con el dataset actual; nuestra red tiene un bias bajo pero sufre un poco de variance (overfitting): gestiona correctamente los datos de validación, pero la generalización es un poco pobre.

En este proyecto en el que estudiamos un mapa topológico el problema con la generalización en realidad no es tal. Simplemente es un problema con la superposición de nodos en un entorno dinámico como podíamos ver en el capítulo 6.4 o en la Figura 7.1 donde se muestran tres nodos del mapa topológico entre los que la red neuronal duda donde clasificar el frame.

Nuestro red neuronal ha sido entrenada con imágenes en las que no contemplábamos superposición de nodos y para resolver este problema y aumentar la tasa



Figura 7.1: Superposición de tres nodos en las pruebas test. La red neuronal duda entre cual de los tres clasificar el frame

de aciertos en test deberíamos tomar más imágenes para el dataset en las que se encontrasen con más de un landmark característico en la imagen. De esta manera, el clasificador sabría como gestionar una imagen donde varios landmark aparecieran al mismo tiempo.

No obstante esto es un arma de doble filo; el realizar imágenes en las que varios landmark aparecieran en una imagen podría confundir a la red neuronal empeorando su tasa de aciertos tanto en validation como en test. Para mejorar esta tasa de aciertos sería más recomendable usar algunas de las técnicas que describiremos en el siguiente Capítulo 7.2

## 7.2. Otras líneas de trabajo

### 7.2.1. Bolsa de palabras visuales

Cambiando completamente de paradigma se podría conjugar un clasificador deep learning más la técnica de bolsa de palabras visuales [11].

Se trata de una técnica precisa pero que requiere de mucho trabajo; la técnica consiste en seleccionar fragmentos relevantes de una imagen (*local patches*) y almacenarlos organizados de manera dividida por clases. El clasificador después buscaría estos fragmentos en las imágenes y asignaría un tanto por ciento de probabilidad de

que la imagen perteneciera a cada posible clase. Este funcionamiento puede verse representado en la Figura 7.2.

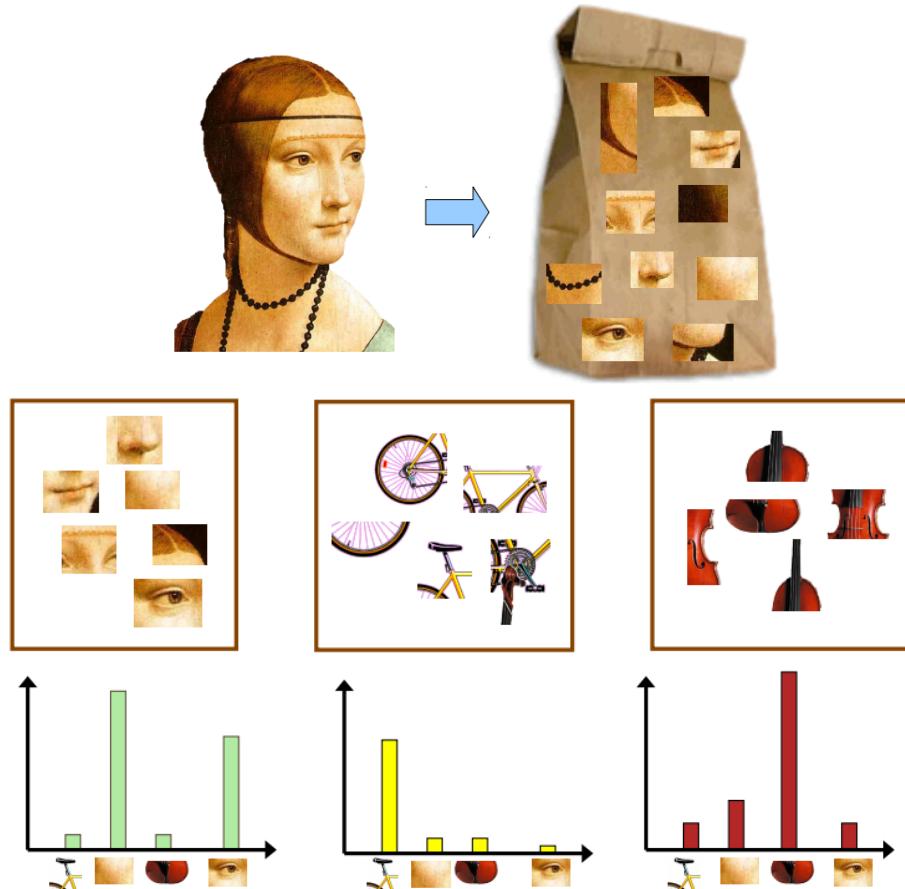


Figura 7.2: Funcionamiento de la bolsa de palabras (Imágenes pertenecientes a la asignatura de *Visión por computador* de este mismo máster)

El único problema con este enfoque es el arduo trabajo de analizar fotogramas buscando fragmentos de imagen relevantes. En un entorno pequeño y con pocas imágenes esto podría realizarse en poco tiempo, pero en un entorno grande muy rico en objetos de distintos tipos como el presentado en este proyecto podría suponer un problema.

### 7.2.2. R-CNN o YoLo

Actualmente las redes que capturan regiones de interés como las R-CNN [10] o sus versiones mejoradas Fast R-CNN o familias de estas [9] se han vuelto muy populares, sobretodo debido a la promesa de los coches autónomos en las próximas décadas o al avance de las técnicas de visión por computador, ya que conocer que una imagen tiene un objeto es útil, pero saber la posición exacta de ese objeto en la imagen es más útil aún.

Mediante esta técnica lo que podríamos hacer es dejar de entrenar la red para que reconociera los nodos como clases y empezaríamos ha entrenar la red para que reconociera objetos individuales del entorno: una maquina expendedora, una losa de piedra, una ventana, etc. Esto podría verse como la representación mostrada en la Figura 7.3.



Figura 7.3: Representación de lo que veríamos con una red R-CNN o YoLo

De esta manera la red nos devolvería una serie de objetos que se encuentran en la escena y, poniendo todos estos objetos en conjunto, podríamos programar que dado que un objeto de la escena se encuentre o no presente se devuelva un tanto por ciento de posibilidades de que la red esté mirando una imagen de un nodo en particular.

El único problema es que estas redes [10, 9] son muy lentas, tardando de 50 a 2 segundos en dar una respuesta, lo que impediría implementar dicha solución para el sistema de navegación en tiempo real de un robot. Es por eso por lo que se debería pensar en usar una red YOLO (You Only Look Once) [8] que son mucho más rápidas, pudiendo ejecutar videos en tiempo real a 40 frames/segundo. Estas redes YOLO tienen la desventaja de tener menor precisión que algunas de las redes familia de R-CNN. aunque las últimas versiones de YOLO presentan aumentos en su precisión que las hacen idóneas para este enfoque encontrando un equilibrio entre rapidez y precisión.

# Apéndice A

## Training completo

Debido a que los datos completos del entrenamiento son demasiado grandes como para ser incluidos en esta tesis los he añadido en estos apéndices.

Tabla A.1: Datos del entrenamiento completo de la red

Epoch	Loss	Accuracy	Val-Loss	Val-Accuracy	Tiempo (segundos)
1	1,1183	0,5995	1,0262	0,6968	14260
2	0,9144	0,6698	3,4629	0,352	221
3	0,4787	0,839	1,5481	0,6467	219
4	0,3493	0,8906	1,1629	0,6619	217
5	0,2942	0,9106	0,7883	0,7449	216
6	0,2347	0,9331	0,8077	0,7824	214
7	0,2056	0,9434	0,2193	0,9335	216
8	0,203	0,9462	0,4999	0,8498	215
9	0,1772	0,956	0,461	0,8617	215
10	0,1678	0,9593	0,2026	0,9409	214
11	0,1667	0,959	0,9479	0,7557	215
12	0,1497	0,9667	0,1069	0,9829	215
13	0,1501	0,9658	0,4005	0,9161	221
14	0,1546	0,9668	0,136	0,973	216
15	0,1487	0,9689	0,1152	0,9813	217
16	0,1416	0,9721	0,2202	0,9416	217
17	0,1296	0,9759	0,174	0,9627	215
18	0,1516	0,968	0,5211	0,8505	215
19	0,1285	0,9778	0,1822	0,955	217
20	0,1228	0,9788	1,919	0,6602	215
21	0,1323	0,976	0,1321	0,9763	215
22	0,1272	0,9782	0,3269	0,926	217
23	0,1266	0,9776	0,1297	0,9741	223
24	0,126	0,9787	0,7721	0,8162	218
25	0,1275	0,9783	0,3253	0,9221	217

Tabla A.2: Datos del entrenamiento completo de la red

Epoch	Loss	Accuracy	Val-Loss	Val-Accuracy	Tiempo (segundos)
26	0,1267	0,9792	0,3894	0,9095	219
27	0,1196	0,9809	0,8695	0,7701	217
28	0,1145	0,9822	0,3764	0,9049	217
29	0,1146	0,9822	0,1773	0,9655	218
30	0,1191	0,9812	0,1151	0,9827	217
31	0,116	0,9829	0,1489	0,9715	217
32	0,1179	0,9818	0,3197	0,9148	217
33	0,112	0,9842	0,61	0,8413	220
34	0,106	0,9857	0,2368	0,9528	217
35	0,1138	0,9828	0,1282	0,9789	217
36	0,1046	0,9849	0,4965	0,8953	214
37	0,1131	0,983	0,6393	0,8422	215
38	0,1033	0,9863	0,4503	0,8742	217
39	0,1047	0,9849	0,1476	0,9717	214
40	0,105	0,9845	0,4035	0,8764	217
41	0,113	0,9822	0,1086	0,9835	216
42	0,092	0,9891	0,4548	0,9188	214
43	0,1002	0,9865	0,2538	0,9377	219
44	0,0943	0,9882	0,228	0,9464	215
45	0,1049	0,9844	0,0944	0,9889	215
46	0,099	0,9863	0,4068	0,8814	214
47	0,0967	0,9871	0,2143	0,9434	214
48	0,0949	0,9879	0,1595	0,9653	215
49	0,0908	0,9887	0,1133	0,98	214
50	0,0973	0,9863	0,0864	0,987	215
51	0,093	0,9876	0,573	0,8735	215
52	0,0881	0,9893	0,8778	0,8068	214
53	0,0887	0,989	0,2969	0,9216	215
54	0,0924	0,9879	0,2079	0,9541	215

Tabla A.3: Datos del entrenamiento completo de la red

Epoch	Loss	Accuracy	Val-Loss	Val-Accuracy	Tiempo (segundos)
55	0,0883	0,9889	0,2514	0,9398	213
56	0,0882	0,9893	0,1453	0,9677	212
57	0,0834	0,9909	0,2007	0,9554	214
58	0,0852	0,989	1,649	0,7137	213
59	0,0891	0,9887	0,1822	0,9563	212
60	0,0873	0,9894	0,3357	0,9153	215
61	0,0829	0,9906	0,3339	0,9168	212
62	0,0868	0,9891	0,502	0,9109	214
63	0,0841	0,99	0,0948	0,9868	219
64	0,0818	0,991	0,9215	0,8151	216
65	0,0866	0,989	0,117	0,9798	212
66	0,08	0,9912	0,335	0,9175	219
67	0,08	0,9908	0,6315	0,8766	215
68	0,0861	0,9889	0,2238	0,9502	212
69	0,0843	0,9904	0,3372	0,9111	30878
70	0,075	0,9923	0,0837	0,9887	222
71	0,076	0,9926	0,1185	0,9798	218
72	0,0751	0,9925	0,1867	0,9557	215
73	0,0841	0,9891	0,1607	0,9679	213
74	0,0744	0,9932	0,0766	0,9901	210
75	0,0779	0,9911	0,1305	0,9761	213
76	0,0741	0,9917	0,0777	0,9901	211
77	0,0685	0,9939	0,2093	0,9486	209
78	0,0792	0,9903	0,0795	0,9875	211
79	0,0745	0,9919	0,2116	0,9502	211
80	0,0751	0,9922	0,1366	0,9728	208
81	0,0756	0,9919	0,4593	0,9207	216
82	0,0767	0,9917	0,1276	0,9769	210
83	0,0752	0,9916	0,0772	0,9901	207
84	0,0698	0,9937	0,0879	0,9849	209
85	0,0868	0,9875	0,0599	0,9982	209
86	0,0695	0,9942	0,0706	0,9932	208
87	0,0727	0,9922	0,095	0,9866	207
88	0,0765	0,991	0,7263	0,8562	207
89	0,069	0,9937	0,1127	0,9794	209
90	0,0654	0,9943	0,068	0,9923	204



# Bibliografía

- [1] Diederik P Kingma and Jimmy Ba. *Adam: A method for stochastic optimization.* arXiv preprint arXiv:1412.6980, 2014.
- [2] Maravall D, de Lope J and Fuentes JP *Vision-based anticipatory controller for the autonomous navigation of an UAV using artificial neural networks*, Neurocomputing, Volume 151, Part 1, 2015, Pages 101-107, ISSN 0925-2312.
- [3] Maravall D, de Lope J and Fuentes JP (2017) *Navigation and Self-Semantic Location of Drones in Indoor Environments by Combining the Visual Bug Algorithm and Entropy-Based Vision*. Front. Neurorobot. 11:46. doi: 10.3389/fnbot.2017.00046
- [4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research 15 (2014) 1929-1958
- [5] Yarin Gal, Zoubin Ghahramani; *Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference*; Conference paper at ICLR 2016
- [6] Krizhevsky, Alex & Sutskever, Ilya & E. Hinton, Geoffrey. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. Neural Information Processing Systems. 25. 10.1145/3065386.
- [7] K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition*. In CVPR, 2016.
- [8] Redmon J, Divvala S, Girshick R, et al. *You Only Look Once: Unified, Real-Time Object Detection*. In CVPR, 2016.
- [9] Ren, S.; He, K.; Girshick, R.; Sun, J. *Faster R-CNN: Towards real-time object detection with region proposal networks*. IEEE Trans. Pattern Anal. Mach. Intell. 2017, 6, 1137–1149.
- [10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation*. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '14). IEEE Computer Society, Washington, DC, USA, 580–587. DOI: <https://doi.org/10.1109/CVPR.2014.81>

- [11] Y. Zhang, R. Jin, and Z.-H. Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.
- [12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. CoRR abs/1511.07289. <http://arxiv.org/abs/1511.07289>.
- [13] Sergey Ioffe and Christian Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. arXiv preprint arXiv:1502.03167, 2015.