Capítulo 3

Generación de imágenes

Resumen: Para dotar a la aplicación de las capacidades de reconocimiento de imágenes es necesario pasar por un proceso previo de entrenamiento con cientos o miles de de ellas correctamente etiquetadas. La obtención de un conjunto de entrenamiento con las características requeridas es una tarea complicada. En este capítulo se describe una opción alternativa que permite generar esas imágenes de forma sintética a partir de modelos tridimensionales.

3.1. Introducción

El objetivo del proyecto es reconocer e identificar distintos desechos y su material principal, e indicar cómo deben reciclarse adecuadamente. Para ello, se utilizan técnicas de visión artificial, lo que conlleva el entrenamiento de una red neuronal. Para la identificación hay que usar redes supervisadas. Eso significa que se requieren fotos correctamente etiquetadas, indicando lo que contienen, para que el sistema aprenda a reconocerlas. Ese entrenamiento es crítico para que todo funcione, y para ello se necesita un gran número de imágenes

Conseguir tantas imágenes, distintas y claras, supone un trabajo costoso y lento; sin tener en cuenta el almacenaje de estas. Contamos con varias opciones para conseguir las imágenes. Una de ellas, y quizá la más obvia, es descargarlas de la red. Esta opción dista de ser la ideal debido a lo tedioso que resulta este procedimiento; buscar, seleccionar y descargar una a una miles de imágenes que cuenten con una buena resolución, es un proceso muy lento.

Otra forma es realizar las fotografías personalmente. Esta cuenta con varios problemas, el primero es que se necesita tener acceso a los objetos, y cuando se necesita hacer miles de imágenes distintas el número de objetos a fotografía es muy alto.

Una posible solución para este problema es acudir a establecimientos donde los vendan y llevar a cabo allí las fotos. Uno de los inconvenientes es que obtener las imágenes llevaría una cantidad de tiempo excesiva. Una alternativa puede ser la grabación de vídeos, utilizando los *frames* como imágenes de entrenamiento. Esta opción es más dinámica frente a las fotografías, pero por otro lado todos los *frames* obtenidos tendrían que ser revisados con atención. Esto es necesario por si aparecen otros objetos además del deseado, ya que se deberían separar para el correcto etiquetado; o, incluso, si ni siquiera aparece. {**TODO TODO TODO:** revisar lo de los objetos ya usados}

En ambas opciones se presentan varios inconvenientes. Los más importantes son, que al realizarse en un establecimiento, existe el riesgo de violar la protección de datos de los clientes y los trabajadores de alrededor, además de que no esté permitido realizar fotografías ni grabaciones dentro del comercio. Otro factor menos significativo es que al tratarse de una aplicación de reciclaje sería preferible que se trate de objetos ya usados, como una lata abierta, una botella vacía o un papel arrugado.

Los avances en aprendizaje automático durante los últimos años han permitido que se convierta en un campo más accesible para interesados con diferentes niveles de conocimiento. Esto ha provocado un gran aumento de materiales disponibles de manera libre.

Una posibilidad que se presenta es utilizar *datasets* de libre uso como COCO (Common Objects in Context) [28]. Aunque este resulta no ser tampoco la opción perfecta, es un acercamiento a una posible solución. Estos *datasets* tan amplios cuentan con tal diversidad de objetivos para identificar que llegan a tener decenas de miles de imágenes. Para poder utilizarlo en este proyecto sería necesario examinarlo entero y seleccionar y separar aquellas que se pudieran utilizar. Esto, como en los casos anteriores, es una tarea sumamente lenta sin ninguna garantía de obtener imágenes suficientes.

Igual que existen *datasets* tan amplios, también los hay pequeños y enfocados a la detección de uno o pocos objetos. A pesar de la limitación presente al tener que encontrar *datasets* específicos para cada objeto que se quiera identificar, resulta la opción más accesible entre las mencionadas.

Otro ámbito en el que se ha conseguido una gran mejoría en las últimas décadas es en los gráficos de contenidos digitales audiovisuales, ya sean películas, videojuegos, videoclips, anuncios, etc. Esta mejora llega hasta tal punto que a veces puede resultar difícil distinguir entre realidad y CGI (*Computer-generated imagery*). Teniendo en cuenta esto, las dificultades mencionadas en la obtención de imágenes y las oportunidades que ofrece la automatización, no se ha querido desaprovechar la posibilidad de enfocarlo desde un punto más cercano a este.

Debido a que las cámaras virtuales y las reales son sensores diferentes entrenando solamente con datos sintéticos la red sufre un desplome en la precisión, así que el uso de las imágenes generadas únicamente tampoco es

una opción aceptable. Además, puesto que está demostrado que el uso de datos reales y sintéticos, combinados, mejora el rendimiento de la red frente al uso de datos reales únicamente [42][1] se considera que la mejor opción es utilizar los *datasets* de menor tamaño en conjunto con imágenes generadas.

{**TODO TODO TODO:** mencionar aquí los proyectos que han usado videojuegos o motores o que los mencionan. Enlazarlo con lo de los gráficos de contenidos digitales audiovisuales diciendo que es buena idea..}

3.2. Aplicación

La aplicación de generación de imágenes ha sido desarrollada utilizando el motor de videojuegos Unity. Las características principales de la aplicación son la carga de modelos tridimensionales y la toma de capturas. La aplicación cuenta con un Game Manager que se encarga de la gestión del trascurso de la aplicación y de las variables principales (rutas de directorios, cantidad de imágenes a generar o la extensión de las capturas generadas). Toma este nombre ya que es el que se suele dar en los juegos al gestor global, y se ha mantenido por inercia de uso de Unity.

Al importar los modelos es necesario crear un objeto reutilizable de Unity para trabajar con ellos de manera más sencilla, denominados *prefabs*. Estos se separan y guardan en carpetas según el material al que pertenecen (metal, plástico, vidrio); lo que facilitará posteriormente el guardado de las capturas generadas.

Durante el tiempo de vida de la aplicación se accede a cada una de estas carpetas, de las que se cargan, uno a uno, los *prefabs* contenidos en ellas. En cada frame se toma una captura de la escena, en ella se presenta uno de los modelos, colocado con una posición y rotación aleatorios, diferente en cada captura. De esta forma, a partir de un mismo objeto se obtienen imágenes muy distintas, permitiendo capturarlos desde diversos ángulos y distancias Para que la posición aleatoria generada no quede fuera del campo de visión de la cámara, se establecen unos valores máximos y mínimos para cada eje de coordenadas. Aún así, para prevenir posibles errores, antes de hacer la captura se comprueba que el modelo se encuentra dentro del campo de visión.

Para generar más diversidad en las imágenes el fondo de estas también es generado de manera aleatoria. Con el fin de hacerlo más completo se mezclan varias texturas. Lo componen dos planos superpuestos, separados levemente para que no se combinen, a los que se les asigna una textura aleatoria. Uno de ellos siempre está visible, mientras que el otro sólo se muestra a veces, en esas ocasiones se le asigna una posición y rotación aleatorias (figura 3.1).

Una vez establecido el objeto y el fondo se toma la captura. Estas, igual

[1] the use of the combined data significantly boosts the performance obtained when using the real-world data alone

(a) Captura con un plano de fondo.

(b) Captura con dos planos de fondo.

Figura 3.1: Capturas de ejemplo de los fondos.

que los modelos, se guardan separadas en carpetas según el material. Esto significa que las capturas de un objeto de metal se guardan en una carpeta llamada metal. Como con los modelos, todas las carpetas de los distintos materiales se almacenan en una carpeta raíz.

Puesto que las imágenes generadas van a ser utilizadas para el entrenamiento de una red neuronal supervisada, todas ellas deben de estar correctamente etiquetadas. Como van a entrenarse utilizando TensorFlow Lite, sobre lo que se hablará en capítulos posteriores (**??** y 5), el etiquetado se realiza de manera muy sencilla. Debido a que el objetivo de la aplicación no es identificar todos los objetos presentes en una imagen, sino identificar únicamente el material del objeto principal, no es necesario tener diversos objetos en una imagen y delimitar cada uno con su etiqueta correspondiente. En este caso, el etiquetado se hace por jerarquía de carpetas. Esto quiere decir que hay una carpeta raíz que contiene distintas subcarpetas, una para cada tipo de material identificable por la red; y dentro de cada una de estas carpetas se almacenan todas las imágenes correspondientes. Para que cada captura tenga un nombre único, se guardan usando el nombre del modelo añadiéndole la hora, minutos, segundos y milisegundos del momento en que ha sido tomada. Pueden exportarse en dos formato PNG o formato JPG, mediante la configuración del Game Manager. PNG es un formato sin pérdida que permite imágenes sin fondo. Puesto que esa propiedad no es necesaria en este proyecto, y que los resultados son similares con ambos formatos, es preferible el uso de JPG debido a que los archivos ocupan menos espacio. Esto es importante ya que se necesita un gran número de imágenes para el entrenamiento de la red neuronal y así no se ocupa tanto espacio en la máquina.

3.2.1. Modelos 3D

{**TODO TODO TODO:** teniendo en cuenta que podrían existir complicaciones con la obtención de los modelos se consiguieron datasets de imágenes de objetos para poder hacer pruebas de prototipado. Teniendo en cuenta los materiales que se pudieron obtener (plástico, vidrio y latas) se continuó en esa línea en la obtención de los modelos para tener un dataset pequeño, y poco variado, pero completo con el que poder hacer pruebas sin problema. Como se comentará en el capítulo 5 {**TODO TODO TODO:** referencia a capitulo 5} para el correcto funcionamiento de la aplicación son necesarios al menos tres objetivos a identificar diferentes. } Esta sección está orientada a explicar el proceso de obtención y uso de los modelos tridimensionales.

Como se ha comentado, para generar imágenes sintéticas mediante esta aplicación, se parte de modelos tridimensionales. Para este proyecto todos los modelos seleccionados y utilizados son de libre uso conseguidos desde varias páginas dedicadas a esto, principalmente CGTrader[2], Free3D[3] y la propia Asset Store de Unity[4]. En esta última opción no se encontraron demasiados recursos para el desarrollo, pero fueron los que se utilizaron en el inicio del proceso; posteriormente la cantidad de modelos se fue aumentado visitando las páginas mencionadas. El formato principal de que se ha tratado de conseguir para los modelos es FBX.

FBX es un formato de archivo propiedad de Autodesk, permite que estos recursos tridimensionales tengan la mínima pérdida de datos entre los distintos softwares de edición 3D[5]. Se decidió utilizar este formato ya que es uno de los más extendidos, se encuentran bastantes recursos sin problema y su carga en Unity suele ser sencilla. Además, ha sido un formato utilizado para modelos de proyectos anteriores por lo que trabajar con ello generaba más tranquilidad que con algo desconocido.

Se llegó a la conclusión de que el problema a veces era que la carga de los materiales y texturas no se hacía correctamente o bien que el modelo no estaba bien exportado y fallaba al importarlo en Unity. Se hicieron numerosas pruebas tratando de abrirlos desde distintos editores para intentar comprender qué ocurría con las texturas. Durante esta exploración se encontró que algunos de estos modelos incluso presentaban problemas mayores. Un ejemplo fue el modelo de una botella de plástico cuyas caras internas se renderizaran al revés, en vez de ver las caras externas {**TODO TODO TODO:** poner imágenes}.

Estos modelos se importaron primero en Blender, y se volvieron a exportar los modelos a FBX pero cambiando la configuración. En algunos casos se aprovechó que el mismo recurso estaba en varios formatos y se partió de

[2] `https://www.cgtrader.com/`
[3] `https://free3d.com/es/`
[4] `https://assetstore.unity.com/`
[5] `https://www.autodesk.com/products/fbx/overview#intro`

otro que no fuera FBX.

3.2.2. Capturas

{**TODO TODO TODO:** cómo se nombran las capturas y que habría que revisar para los objetos grandes y pequeños.}

Como se explicó en el apartado del desarrollo, para realizar capturas se programó un script que generara las imágenes a partir de la cámara de la escena de Unity.

Para llevarlo a cabo se utilizó como referencia un tutorial de Code Monkey [6]. Para generar las imágenes se tienen dos métodos principales. El primero establece que se quiere guardar una captura, el tamaño de esta, el directorio donde se guardará y si es en formato PNG o JPG.

El otro método, se encarga de la lógica de generar la imagen, para ello se ha utilizado el método de MonoBehaviour "OnPostRender()"[7]. Este método se llama siempre después de que la cámara renderice la escena y ejecuta el código que se haya introducido. Para asegurarse de que no generaba imágenes en frames que no se requería, el código de este método se regula mediante un booleano. Todas las variables que este método necesita son asignadas en el método mencionado anteriormente; con esa información se permite que el código de generar capturas se ejecute y como resultado guarde la captura del tamaño y formato establecido, en el directorio indicado.

Para no perder información, el tamaño de las capturas es el mismo que la renderización de la cámara. Como se comentó previamente, las imágenes generadas se guardan dentro de una misma carpeta pero separadas en subdirectorios según a qué material corresponde el objeto de la imagen.

Unity permite guardar las imágenes en dos formatos distintos, JPG[8] y PNG [9]; para este proyecto se decidió usar PNG. Esta elección se tomó por querer evitar la pérdida de información y que fueran lo mejor posible para el entrenamiento. Posteriormente se llegó a la conclusión de que con el formato JPG los resultados habrían sido similares pero ocupando menos espacio en el disco. En cualquier caso, el cambio de formato se hace de manera muy sencilla, es una casilla que se activa y desactiva en los componentes del GameManager desde el editor de Unity.

También, en los componentes del GameManager, se puede elegir la cantidad de imágenes que se quieren tomar de cada objeto. Este valor está puesto por defecto a 10 pero puede ampliarse y reducirse todo lo que se desee, siempre y cuando el valor se un número positivo y entero.

[6] `https://www.youtube.com/watch?v=lT-SRLKUe5k`

[7] `https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnPostRender.html`

[8] JPG: `https://docs.unity3d.com/ScriptReference/ImageConversion.EncodeToJPG.html`

[9] PNG: `https://docs.unity3d.com/ScriptReference/ImageConversion.EncodeToPNG.html`

Bibliografía

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] C. Alfonso, R. Estévez Estévez, J. M. Lobo, B. Lozano Diéguez, F. Prieto, J. Santamarta, and A. Gaerter. Emergencia climática en España. Diciembre 2016.

[3] R. Almond, G. M., and T. Petersen. Wwf (2020) living planet report 2020 - bending the curve of biodiversity loss. *World Wildlife Fund (WWF)*, 2020.

[4] O. Alsing. Mobile object detection using tensorflow lite and transfer learning. Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2018.

[5] Y. Amit. *2D Object Detection and Recognition: Models, Algorithms, and Networks*. Mit Press. MIT Press, 2002.

[6] Y. Amit, P. Felzenszwalb, and R. Girshick. *Object Detection*. Springer International Publishing, Cham, 2020.

[7] X. Basogain Olabe. Redes neuronales artificiales y sus aplicaciones. *Dpto. Ingeniería de Sistemas y Automática, Escuela Superior de Ingeniería Bilbao. Open Course Ware.[En línea] disponible en http://ocw. ehu. es/ensenanzas-tecnicas/redes-neuronales-artificiales-y-sus-aplicaciones/Course_ listing.[Consultada 20-09-2012]*, 2008.

[8] M. Caballero, S. Lozano, and B. Ortega. Efecto invernadero, calentamiento global y cambio climático: una perspectiva desde las ciencias de la tierra. *Revista digital universitaria*, 8, 2007.

[9] J. Cohen, C. F. Crispim-Junior, C. Grange-Faivre, and L. Tougne. CAD-based Learning for Egocentric Object Detection in Industrial Context. In *15th International Conference on Computer Vision Theory and Applications*, volume 5, Valletta, Malta, Feb. 2020. SCITEPRESS - Science and Technology Publications.

[10] G. Cortina Fernández. Técnicas inteligentes para su integración en un vehículo autómata. Trabajo de Fin de Grado en Ingeniería del Software, Facultad de Informática UCM, Departamento de Ingeniería del Software e Inteligencia Artificial, Curso 2019/2020., 2020.

[11] B. Cyganek. *Object Detection and Recognition in Digital Images: Theory and Practice*. Wiley, 2013.

[12] R. Flórez López, J. M. Fernández, and J. M. Fernández Fernández. *Las Redes Neuronales Artificiales*. Metodología y Análisis de Datos en Ciencias Sociales. Netbiblo, 2008.

[13] R. Fonfría, R. Sans, and J. de Pablo Ribas. *Ingeniería ambiental: contaminación y tratamientos*. Colección productiva. Marcombo, 1989.

[14] S. Frintrop. *VOCUS: A visual attention system for object detection and goal-directed search*, volume 3899. Springer, 2006.

[15] L. García Rodríguez. *Algunas cuestiones notables sobre el modelo de Hopfield en optimización*. PhD thesis, Madrid, Noviembre 2018. Tesis de la Universidad Complutense de Madrid, Facultad de Ciencias Matemáticas, Departamento de Estadística e Investigación Operativa, leída el 15-12-2017.

[16] G. A. Gómez Rojas, J. C. Henao López, and H. Salazar Isaza. Entrenamiento de una red neuronal artificial usando el algoritmo simulated annealing. *Scientia Et Technica*, 2004.

[17] G. Guridi Mateos et al. Modelos de redes neuronales recurrentes en clasificación de patentes. B.S. thesis, 2017.

[18] J. R. Hilera and V. J. Martínez Hernando. *Redes neuronales artificiales: fundamentos, modelos y aplicaciones*. 01 1995.

[19] S. Hinterstoisser, S. Benhimane, V. Lepetit, P. Fua, and N. Navab. Simultaneous recognition and homography extraction of local patches with a simple linear classifier. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2008. doi:10.5244/C.22.10.

[20] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[21] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[22] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size, 2016.

[23] A. Iguarán Guerra, S. Gómez Ruíz, et al. Análisis de las necesidades y dificultades en la disposición de residuos sólidos en la fuente doméstica para el desarrollo de un producto. B.S. thesis, Universidad EAFIT, 2010.

[24] Y.-C. Jhang, A. Palmar, B. Li, S. Dhakad, S. K. Vishwakarma, J. Hogins, A. Crespi, C. Kerr, S. Chockalingam, C. Romero, A. Thaman, and S. Ganguly. Training a performant object detection ML model on synthetic data using Unity Perception tools, Sep 2020.

[25] R. Karim. *TensorFlow: Powerful Predictive Analytics with TensorFlow*. Packt Publishing, Limited, 2018.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[27] P. Larranaga, I. Inza, and A. Moujahid. Tema 8. redes neuronales. *Redes Neuronales, U. del P. Vasco*, 12, 1997.

[28] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2015.

[29] P. López and J. García-Consuegra Bleda. *Informática gráfica*, volume 19. Univ de Castilla La Mancha, 1999.

[30] M. A. López Pacheco. Identificación de sistemas no lineales con redes neuronales convolucionales. *Cuidad de Mexico: Centro de investigación y de estudios avanzados*, 2017.

[31] D. J. Matich. Redes neuronales: Conceptos básicos y aplicaciones. *Universidad Tecnológica Nacional, México*, 41, 2001.

[32] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), 1943.

[33] A. J. McMichael, D. Campbell-Lendrum, S. Kovats, S. Edwards, P. Wilkinson, T. Wilson, R. Nicholls, S. Hales, F. Tanser, D. L. Sueur, M. Schlesinger, and N. Andronova. Chapter 20 global climate change.

[34] M. Minsky and S. A. Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[35] S. Morant Gálvez. Desarrollo de un sistema de bajo coste para el análisis de tráfico mediante el uso de deep learning. 2021.

[36] L. Moreno Díaz-Alejo. Análisis comparativo de arquitecturas de redes neuronales para la clasificación de imágenes. Master's thesis, 2020.

[37] B. Müller, J. Reinhardt, and M. Strickland. *Neural Networks: An Introduction*. Physics of Neural Networks. Springer Berlin Heidelberg, 1995.

[38] C. Parra Ramos and D. Regajo Rodríguez. Reconocimiento automático de matrículas. *Universidad Carlos III de Madrid*, 2006.

[39] R. Pavón Benítez. Técnicas de deep learning para el reconocimiento de movimientos corporales. Trabajo de Fin de Grado en Ingeniería del Software, Facultad de Informática UCM, Departamento de Ingeniería de Software e Inteligencia Artificial, Curso 2019/2020, 2020.

[40] A. Polacco and K. Backes. The amazon go concept: Implications, applications, and sustainability. *Journal of Business and Management*, 24(1), 2018.

[41] A. Polacco and K. Backes. The amazon go concept: Implications, applications, and sustainability. *Journal of Business and Management*, 24(1), 2018.

[42] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[43] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[44] A. Rozantsev, V. Lepetit, and P. Fua. On rendering synthetic images for training an object detector. *Computer Vision and Image Understanding*, 137, 11 2014.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088), 1986.

[46] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[47] R. Salas. Redes neuronales artificiales. *Universidad de Valparaíso. Departamento de Computación*, 1, 2004.

[48] M. Sánchez and J. Castro. *Gestión y Minimización de Residuos*. Fundación Confemetal, 2007.

[49] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[50] O. Simeone. A very brief introduction to machine learning with applications to communication systems. *IEEE Transactions on Cognitive Communications and Networking*, 4(4), 2018.

[51] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, June 2015.

[53] A. Terceño Ortega. Análisis de un modelo predictivo basado en google cloud y tensorflow. Trabajo de Fin de Grado en Ingeniería Informática y Matemáticas (Universidad Complutense, Facultad de Informática, curso 2016/2017), 2017.

[54] Unity Technologies. Unity Perception package, 2020.

[55] S.-C. Wang. *Artificial Neural Network*. Springer US, Boston, MA, 2003.

[56] P. Warden and D. Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2019.

[57] W. Yu and Y. Bai. Visualizing and comparing alexnet and vgg using deconvolutional layers. 2016.

[58] J. Zamorano Ruiz et al. Comparación y análisis de métodos de clasificación con las bibliotecas scikit-learn y tensorflow en python. 2019.

[59] J. Zurada. *Introduction to Artificial Neural Systems*. West, 1992.