Capítulo 4

Red Neuronal

Resumen: En este capítulo explican las características de la red neuronal, necesaria para llevar a cabo la identificación de objetos. Tensorflow Lite ha sido el framework seleccionado y a partir del cual se ha desarrollado el script de entrenamiento. Para elegir la mejor relación entre imágenes sintéticas y reales que utilizar para la aplicación, se ha realizado una investigación probando con diversas proporciones y comparando los resultados.

4.1. Introducción

Para llevar a cabo la identificación de objetos necesaria para el funcionamiento de la aplicación de este trabajo, es necesario utilizar técnicas y herramientas de aprendizaje automático.

El aprendizaje automático es un subcampo de las ciencias de la computación y una rama de la inteligencia artificial. Tiene como objetivo desarrollar técnicas que permitan que las computadoras aprendan[1]. Para aplicar esto a los distintos campos en los que es aprovechado, se trabaja con redes neuronales. Las redes neuronales artificiales simulan el funcionamiento de las biológicas, cuentan con un conjunto de unidades denominadas neuronas conectadas entre sí las cuales se transmiten señales[2]. {**TODO TODO TODO:** buscar referencias más "formales"}

Esas redes neuronales deben entrenarse a partir de un conjunto de datos amplio para conseguir el comportamiento buscado. Tras finalizar el entrenamiento la red obtiene un valor que indica la precisión con la que puede detectar aquello para lo que se le ha entrenado.

Como conjunto de datos para el entrenamiento de la red para este proyecto se utilizan las imágenes generadas en el capítulo anterior (capítulo 3).

[1] `https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico`
[2] `https://es.wikipedia.org/wiki/Red_neuronal_artificial`

Buscando el funcionamiento óptimo de la aplicación se probarán distintas configuraciones mezclando imágenes reales con sintéticas.

El desarrollo, a nivel de programación, de la red neuronal se lleva a cabo partiendo de los ejemplos disponibles de TensorFlow Lite, para facilitar la realización de la lógica necesaria para generar y entrenar la red neuronal.

4.2. TensorFlow y TensorFlow Lite

El equipo de Google Brain comenzó a investigar en 2011 el uso de redes neuronales a gran escala para utilizarlo en los productos de la empresa. Como resultado desarrollaron DistBelief, su primer sistema de entrenamiento e inferencia escalable. A partir de este, basándose en la experiencia y en un entendimiento más completo de las necesidades y requerimientos del sistema ideal, desarrollaron TensorFlow[3]. Este utiliza modelos de flujo de datos y los *mapea* en una gran variedad de diferentes plataformas, desde dispositivos móviles, como máquinas sencillas de una o varias GPUs, hasta sistemas a gran escala de cientos de máquinas especializadas con miles de GPUs. La API de TensorFlow y las implementaciones de referencia fueron lanzadas como un paquete de código abierto bajo una licencia de Apache 2.0 en noviembre de 2015; (Abadi, Agarwal, Barham, Brevdo, Chen, Citro, Corrado, Davis, Dean, Devin, Ghemawat, Goodfellow, Harp, Irving, Isard, Jia, Jozefowicz, Kaiser, Kudlur, Levenberg, Mané, Monga, Moore, Murray, Olah, Schuster, Shlens, Steiner, Sutskever, Talwar, Tucker, Vanhoucke, Vasudevan, Viégas, Vinyals, Warden, Wattenberg, Wicke, Yu y Zheng, 2015).

Como el nombre de TesorFlow indica, las operaciones son llevadas a cabo por redes neuronales en *arrays* multidimensionales de datos, mejor conocidos como tensores. Puesto que la estructura son grafos de flujo de datos, en estos, los nodos corresponden a las operaciones matemáticas, como sumas, multiplicaciones, factorización y demás; en cambio, los bordes corresponden a los tensores. (Karim, 2018).

Como se ha comentado, TensorFlow fue desarrollado para ser ejecutado en sistemas muy diversos, incluidos dispositivos móviles. Este fue llamado TensorFlow Mobile y permitió a los desarrolladores para móvil crear aplicaciones interactivas sin los retrasos que generaban los cálculos computacionales de aprendizaje automático.

A pesar de las optimizaciones para mejorar la actuación de los modelos y que el mínimo *hardware* requerido era bastante accesible; seguía existiendo cuello de botella en la velocidad de cálculo computacional por la baja latencia de los dispositivos móviles. Por ejemplo, un dispositivo móvil cuyo *hardware* era capaz de ejecutar 10 GFLOPS[4] estaba limitado a ejecutar un modelo de

[3] `https://www.tensorflow.org/`

[4] *Giga floating point operations per second*, proveniente de FLOPS, operaciones de coma flotante por segundo.

5 GFLOPS a 2 4FPS[5], lo que provocaba que las aplicaciones no funcionaran como era esperado (Alsing, 2018).

TensorFlow Lite es la evolución de TensorFlow Mobile, algunas de las optimizaciones que incluye son el uso de *frameworks* como la API de redes neuronales de Android y redes neuronales optimizadas para móvil como MobileNets (Howard, Zhu, Chen, Kalenichenko, Wang, Weyand, Andreetto y Adam, 2017) y SqueezeNet(Iandola, Han, Moskewicz, Ashraf, Dally y Keutzer, 2016).

TensorFlow Lite permite ejecutar modelos de aprendizaje profundo en dispositivos móviles. Estos son entrenados en una computadora y posteriormente trasladados al dispositivo sin necesidad de utilizar un servidor. Utiliza MobileNet, la cual está diseñada y optimizada para imágenes en móviles, incluyendo detección y clasificación de objetos, detección de caras y reconocimiento de lugares. Los modelos de TensorFlow Lite generados en el entrenamiento tienen como extensión de archivo *.tflite* el cual tiene formato FlatBuffer. Tensorflow Lite no se limita a los modelos que han sido directamente creados con esta extensión, sino que en la documentación de TensorFlow se encuentra un conversor que toma un modelo en otro formato y lo convierte a *.tflite*[6].

Fue el framework elegido puesto que cumplía con todas las necesidades que requería el proyecto para el entrenamiento de la red neuronal. Con una amplia documentación y numerosos ejemplos de uso facilitando el trabajo. Además, otro factor importante es el fácil traspaso del resultado a una aplicación móvil sobre la que se hablará en profundidad en el capítulo 5, y el uso del modelo en ella sin necesidad de servidores intermedios como se contará. Asimismo la agilidad del trabajo al tratarse de plataformas en las que se tiene experiencia. {**TODO TODO TODO:** Volver sobre este párrafo}

4.3. Entrenamiento

La fase de entrenamiento del proyecto está basada en las recomendaciones para generar modelos de TensorFlow Lite, ya que el modelo que se va a utilizar para la identificación es el de este formato. Para llevar a cabo el entrenamiento de la red neuronal se ha realizado un *script* que se encargue de la lógica de esto. Este *script* está basado en el ejemplo de Tensorflow Lite y puede usarse desde Google Collab, donde está disponible para entrenar los modelos que se desee[7]. Este utiliza la librería Task de TensorFlow Lite, la cual contiene herramientas para que los desarrolladores creen experiencias de aprendizaje automático con Tensorflow Lite, tales como un clasificador

[5] *frames* por segundo, del inglés *frames per second*.

[6] `https://www.tensorflow.org/lite/convert?hl=es-419`

[7] `https://colab.research.google.com/drive/1sqBewUnvdATOO-yblj55EBFb2sM24XHR?hl=es-419`

de imágenes, detector de objetos o clasificador de lenguaje natural, entre otras. También la biblioteca Model Maker, la cual simplifica el proceso de adaptación y conversión de un modelo de red neuronal de TensorFlow para aplicaciones de aprendizaje automático. Y por último, MobileNetV2(Sandler, Howard, Zhu, Zhmoginov y Chen, 2018), que utiliza convolución separable en profundidad haciendo la red más eficiente.

Para este proyecto se desarrolló el *script* en el dispositivo para así evitar tener que cargar tantas imágenes a la plataforma de Google Collab y descargar el modelo constantemente para utilizarlo. Basándose en el que proporciona TensorFlow de ejemplo, el archivo resulta bastante similar pero con unos cambios personalizados. El ejemplo está planteado para cargar todas las imágenes y después separarlas en entrenamiento y *test*, cada grupo tendrá el porcentaje de imágenes que se le establezca. Por ejemplo, por defecto se reparten en el 90 % de las imágenes para entrenamiento y el 10 % restante para validación. Pero esta no es la funcionalidad requerida para este proyecto. Puesto que el entrenamiento se va a realizar a partir de imágenes generadas y posteriormente se va a utilizar sobre objetos reales, para comprobar la viabilidad del modelo es necesario que las imágenes de *test* sean imágenes reales. Por lo tanto, en este caso, es necesario cargar dos carpetas diferentes para los dos grupos. Puesto que estas siguen teniendo que estar separadas por el material al que pertenecen(plástico, metal, vidrio, etc.) dentro de las dos carpetas para entrenamiento y *test* se encuentran las imágenes separadas en subcarpetas según esta distribución. Para mayor claridad, la estructura de carpetas queda como se representa en la figura 4.1. {**TODO TODO TODO:** Poner una foto más decente, por favor} En esta separación se ha mantenido la proporción aproximada de 90 % de imágenes de entrenamiento y 10 % de validación para todos los materiales.

El entrenamiento se divide en varios *epochs*. Los *epochs* son el número de veces que el algoritmo recorre todos los datos. Estos datos se dividen en *batchs*, y en cada uno de estos se recoge una pequeña parte de los datos. Cada *batch* se recorre en una iteración por lo que cada *epoch* tiene tantas iteraciones como hagan falta para recorrer al completo el conjunto de datos. Esto resulta en que en cada *epoch* el número de iteraciones es el resultado de dividir la cantidad de datos entre el tamaño de *batch*, (Warden y Situnayake, 2019). Por ejemplo, si se tienen 2400 imágenes y el tamaño de *batch* es 32, en cada *epoch* habría 75 iteraciones. Para estos valores se han mantenido los de por defecto de TensorFlow ya que se ha considerado que eran adecuados; siendo el número de *epochs* 5 y el tamaño de *batch* 32.

Con todo lo mencionado se genera el modelo entrenado con extensión *.tflite*, además de un documento de texto plano con las etiquetas de los distintos materiales. Estos dos archivos son lo que se han de trasladar a la aplicación móvil para ser utilizados.

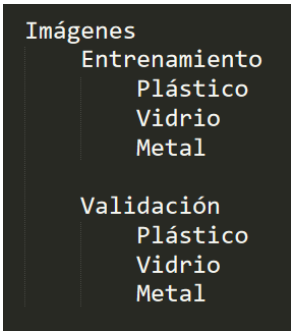

Figura 4.1: Organización de las carpetas con las imágenes separadas.

4.4. Resultados

Con las imágenes sintéticas generadas y el *script* de entrenamiento de la red neuronal programado, se pasa a investigar con qué porcentaje de imágenes sintéticas se obtiene un entrenamiento óptimo de la red que permita el correcto funcionamiento de la aplicació.

Esta investigación se ha llevado a cabo con las opciones para el prototipado de la aplicación debido a las dificultades mencionadas en la sección 3.3.4 con la obtención de modelos tridimensionales. Esto significa que se ha limitado el entrenamiento a utilizar imágenes reales para dos de los tres materiales seleccionados (vidrio y plástico). En cambio, para el material restante (metal) llevar a cabo las pruebas y la comparación de precisión de la red mezclando imágenes reales y las propias generadas. , Se ha seleccionado este material debido a que es del que más modelos tridimensionales se tienen y por lo tanto es para el que más imágenes pueden generarse. El resto de imágenes utilizadas se han obtenido de datasets abiertos al uso público de Kaggle[8].

Para la comparación se ha comenzando con sólo imágenes reales y se han ido aumentando paulatinamente en un 10 % las imágenes generadas hasta contar con un *dataset* de sólo imágenes sintéticas. La figura 4.2 corresponde al crecimiento de la precisión de la red durante el entrenamiento; se encuentra representado para los distintos porcentajes de imágenes generadas y reales. En todos los casos crece de manera similar sin haber grandes diferencias según la proporción de imágenes sintéticas.

En la figura 4.3, por el contrario, se observa cómo varía la precisión al probarse el modelo con los datos de *test*, aquí se sacan las primeras conclusiones sobre la red. Se observa que la mayoría de las opciones se mantienen casi todo el proceso en una precisión mayor del 90 %. Se pueden destacar los datasets con los porcentajes de 0 % y 10 % de imágenes reales cuya precisión se queda muy por debajo de la del resto. La figura 4.4 muestra el valor final

[8] `https://www.kaggle.com/`

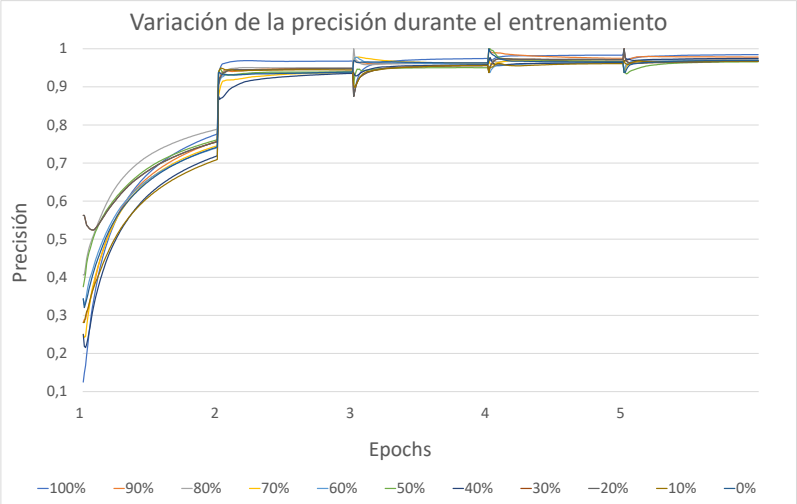

Figura 4.2: Variación de la precisión de la red neuronal a lo largo del entrenamiento de esta.

de la precisión en los distintos casos. A partir de esta última figura pueden sacarse las conclusiones de con qué porcentaje se obtendrían los mejores resultados.

Los dos *datasets* que cuentan con la mayor cantidad de imágenes generadas serían los menos recomendados para utilizar debido a su baja precisión. En cambio, a partir del 80 % de imágenes generadas se observa que la precisión siempre está por encima del 90 %, aunque nunca llega a superar el 95 %.

Con los resultados obtenidos se considera como mejor opción utilizar un 70 % de imágenes generadas para el *dataset* final. Esta opción es de las que mayor precisión tienen y a la vez permite tener un *dataset* que funciona adecuadamente formado principalmente por imágenes sintéticas, lo que facilita la obtención de este.

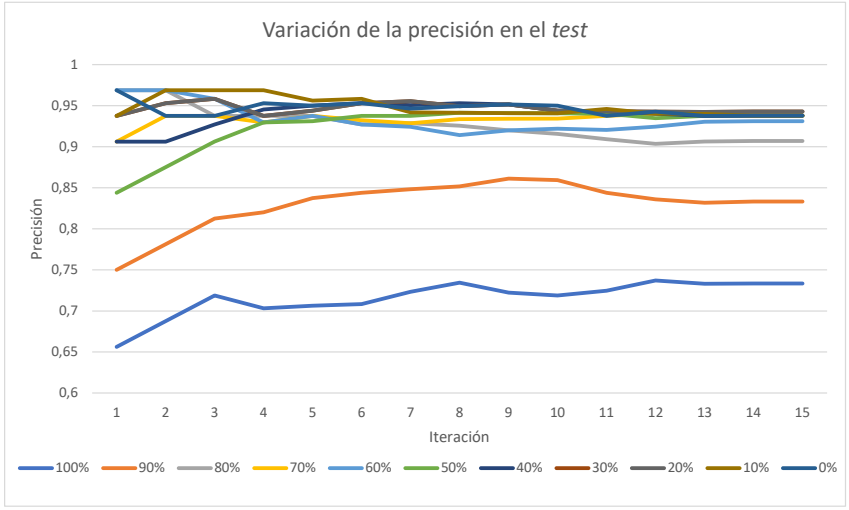

Figura 4.3: Variación de la precisión de la red neuronal durante las pruebas de esta.

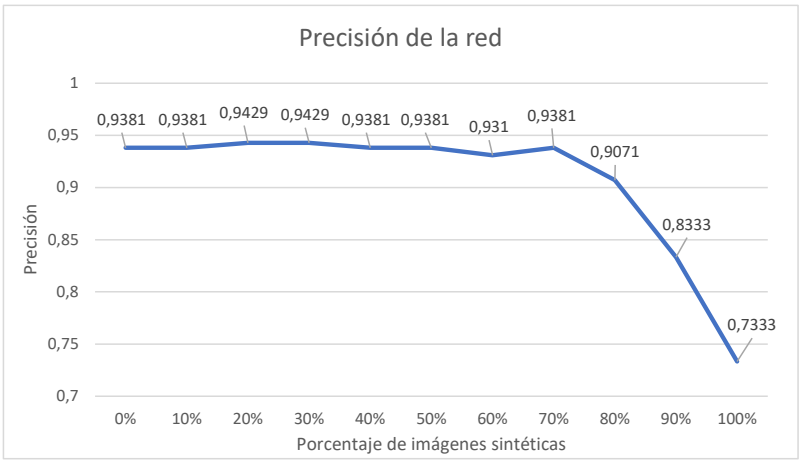

Figura 4.4: Variación de la precisión de la red neuronal según la proporción de imágenes reales y sintéticas.

Bibliografía

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

Alfonso, C., Estévez Estévez, R., Lobo, J. M., Lozano Diéguez, B., Prieto, F., Santamarta, J. y Gaerter, A. Emergencia climática en España. 2016.

Alsing, O. *Mobile Object Detection using TensorFlow Lite and Transfer Learning*. Proyecto Fin de Carrera, KTH, School of Electrical Engineering and Computer Science (EECS), 2018.

Fonfría, R., Sans, R. y de Pablo Ribas, J. *Ingeniería ambiental: contaminación y tratamientos*. Colección productiva. Marcombo, 1989. ISBN 9788426707420.

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

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

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

Sánchez, M. y Castro, J. *Gestión y Minimización de Residuos*. Fundación Confemetal, 2007. ISBN 9788496743342.

SANDLER, M., HOWARD, A., ZHU, M., ZHMOGINOV, A. y CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

WANG, S.-C. *Artificial Neural Network*, páginas 81–100. Springer US, Boston, MA, 2003. ISBN 978-1-4615-0377-4.

WARDEN, P. y SITUNAYAKE, D. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2019. ISBN 9781492051992.