Capítulo 4

Red Neuronal

Resumen:

4.1. 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. 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 GPU hasta sistemas a gran escala de cientos de máquinas especializadas con misles de GPUs. La API de TensorFlow e implementaciones de referencia fueron lanzadas como un paquete de código abierto bajo una licencia de Apache 2.0 en noviembre de 2015; puede encontrarse todo en su página web[1] (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).

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

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 en los modelos para mejorar su actuación, 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 GigaFLOPS estaba limitado a ejecutar un modelo de 5 GFLOPS a 2 FPS, lo que provocaba que la aplicación no funcionara 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).

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 y lo convierte a formato *.tflite*

4.2. Entrenamiento

Para llevar a cabo el entrenamiento de la red neuronal es necesario realizar un *script* que se encargue de la lógica de esto. Este *script*, en primer lugar, carga las imágenes con las que se va a entrenar y validar, en este caso estas están separadas según el material al que pertenecen (plástico, metal, vidrio, etc.). La separación entre imágenes de entrenamiento y de validación se puede realizar de dos maneras: la primera es que todas las imágenes de un mismo material se encuentran en la misma carpeta, como se muestra en la figura 4.1, en este caso separa las imágenes utilizando un 90 % de ellas para entrenamiento y 10 % para validación. En la segunda forma las imágenes se encuentran separadas como se muestra en la figura 4.2. En este caso se ha mantenido aproximadamente las mismas proporciones de 90-10 en la cantidad de las imágenes que en el caso anterior.

La primera opción es útil cuando todas las imágenes son del mismo ti-

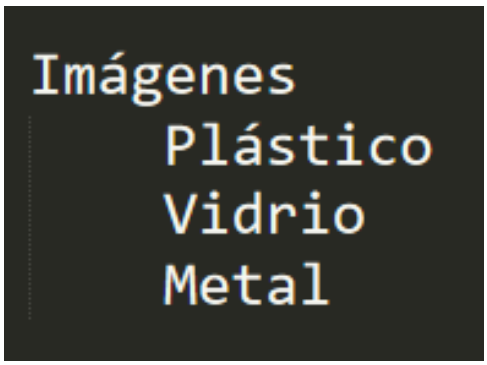

Figura 4.1: Organización de las carpetas con las imágenes unificadas.

po, por ejemplo en las pruebas iniciales del script se probó que funcionaba correctamente utilizando únicamente imágenes reales. Posteriormente, al introducir las imágenes sintéticas, puesto que se entrena con estas pero debe validarse con las reales, se pasó al segundo formato quedando este como definitivo.

Tras llevarse a cabo el entrenamiento se genera el modelo entrenado, este es un archivo con extensión *.tflite*, que pertenece a Tensorflow Lite para que pueda ser utilizado desde dispositivos móviles como se ha comentado previamente {**TODO TODO TODO:** asegurarme de que esto es así}. Además de este archivo, se genera un documento de texto plano con las etiquetas correspondientes a cada material.

4.3. Resultados

{**TODO TODO TODO:** Resultados con cada opción de imágenes. Quizá imágenes de la aplicación haciéndose un lío.}

4.4. Comparación

{**TODO TODO TODO:** [} Comparaciones de cómo de bien/mal sale el resultado]

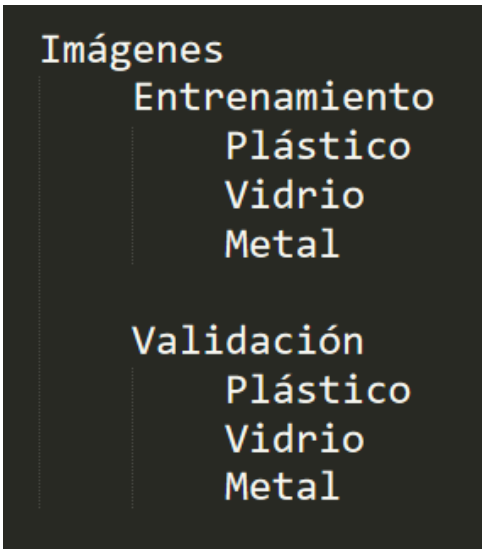

Figura 4.2: Organización de las carpetas con las imágenes separadas.

Bibliografía

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

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.

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.

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.

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