Machine Learning en dispositivos embebidos

Métodos y Técnicas del Aprendizaje Profundo

Francisco Ollero Pacheco

24 de Abril de 2.022

universidad de león

Contenidos

1. Introducción y Objetivo del proyecto.
2. TensorFlow.
3. Modelo.
4. ‘Datasets’.
5. Métricas.
6. Resultados obtenidos.
7. Conclusiones.
8. References.

1. Introducción

ML en dispositivos embebidos : TinyML

TinyML define a sistemas de ML adecuados para dispositivos que tienen memoria y potencia de procesamiento limitadas, y en los que la conectividad a Internet no está presente o es limitada.

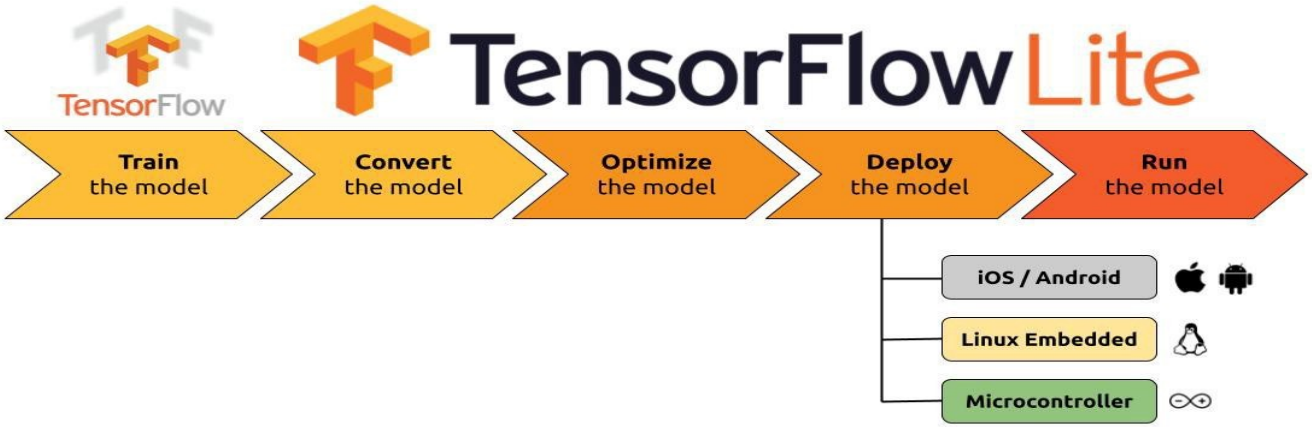
Nos da la posibilidad de incorporar dispositivos ‘inteligentes’ es entornos hasta ahora impensables (entornos naturales, entornos rurales, medicina…).

Para qué y dónde:

https://www.tinyml.org/industry-news/

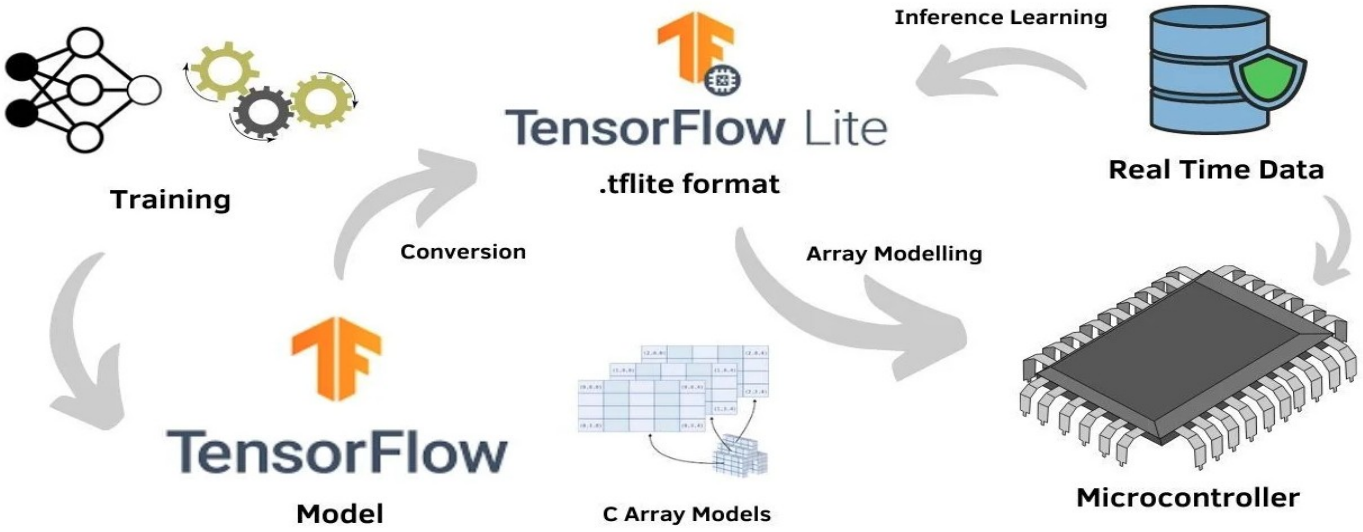
https://www.edgeimpulse.com/solutions/industry

1. Objetivo de este proyecto:



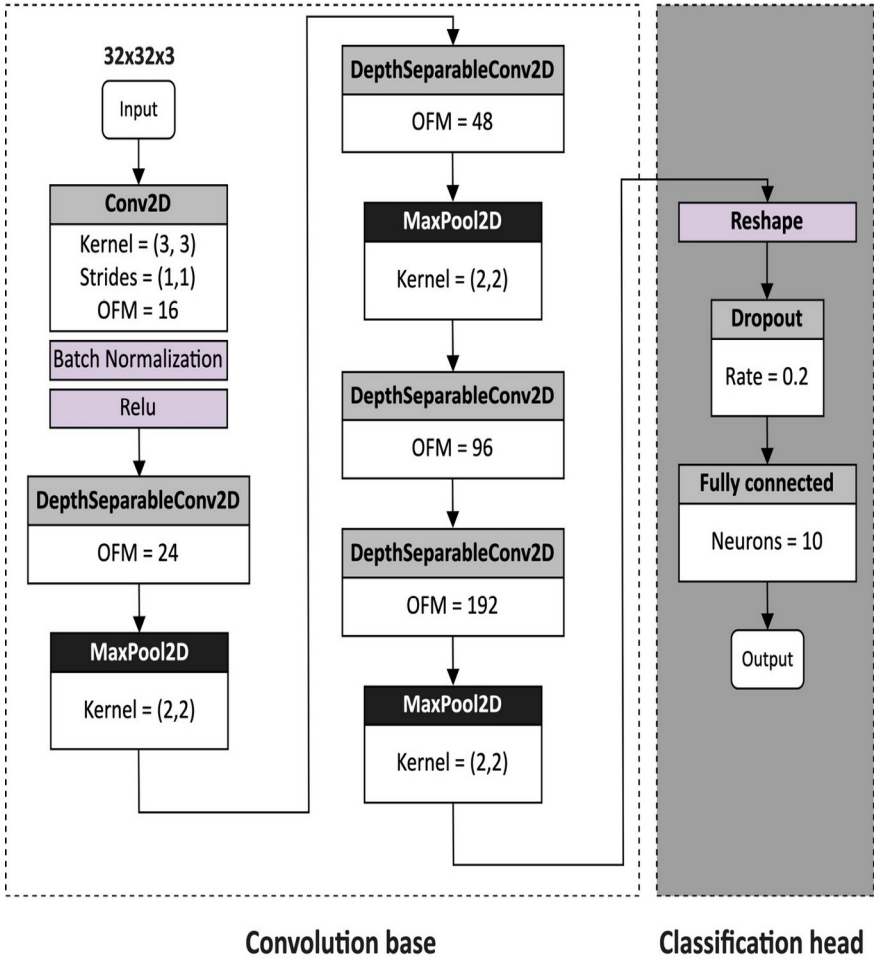
- Crear Modelo de clasificación de imagenes con TF y API Keras, utilizando TF Lite y TF Lite para microcontroladores (TFLu).
- Se tomará como base la estructura de MobileNetV1 (que es el modelo seleccionado) y se utilizará como dataset de entrenamiento CIFAR-10.
- Una vez generado, probado y transformado para su uso en microcontroladores se emulará en un 'Arm Cortex-M3' (con el emulador QEMU con OS Zephyr).

2. Tensorflow:



- TensorFlow Lite es un ‘framework’ de aprendizaje para dispositivos de bajo tamaño y computación.
- Permite ejecutar sus modelos en hardware específico.
- Se permite la conversión del modelo TensorFlow al modelo TensorFlow Lite. Formato .tflite.
- Con optimización y cuantificación de parámetros permite reducir el tamaño y la latencia del modelo no tiene necesidad de conexión a Internet reducen el consumo de energía en el dispositivo.
- TensorFlow Lite Micro : Es una biblioteca de subconjuntos de TensorFlow Lite, conocida como TensorFlow Lite Micro. Ejecuta específicamente modelos de aprendizaje automático en microcontroladores. El desarrollo de TensorFlow Lite Micro se basa en C++ 11, que necesita una arquitectura de 32 bits para la compatibilidad.
- La conversión del archivo .tflite es necesaria a un formato de estructura de matriz, que sea compatible con los microcontroladores.

3. Modelo:<Mobilenet V1>



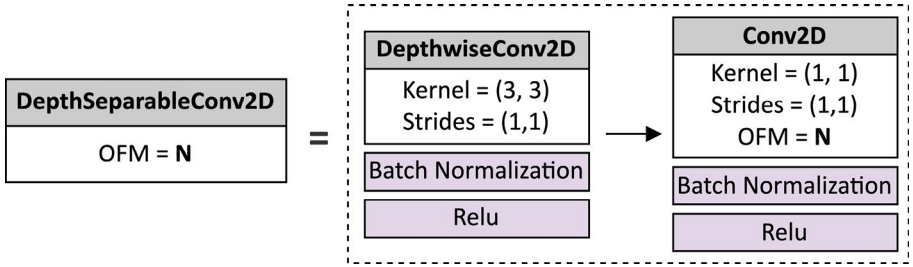
▾ Diseño de la convolución base

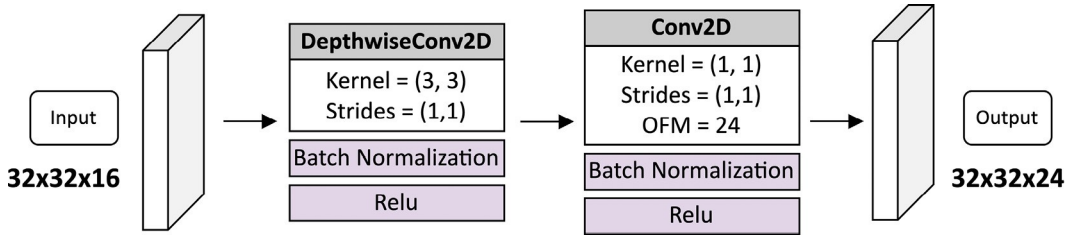
```
[ ] input = layers.Input((32,32,3))

    x = layers.Conv2D(16, (3, 3), padding='same', name='conv1')(input)
    x = layers.BatchNormalization(name='bn1')(x)
    x = layers.Activation("relu", name='act1')(x)
    x = separable_conv(x, 24, 2)
    x = layers.MaxPooling2D((2, 2), name='pool1')(x)
    x = separable_conv(x, 48, 3)
    x = layers.MaxPooling2D((2, 2), name='pool2')(x)
    x = separable_conv(x, 96, 4)
    x = separable_conv(x, 192, 5)
    x = layers.MaxPooling2D((2, 2), name='pool3')(x)
```

▾ Diseño del clasificador

```
[ ] x = layers.Flatten()(x)
    x = layers.Dropout(0.2)(x)
    x = layers.Dense(10)(x)
```

3. Modelo:<Mobilenet V1>(continuación)





▾ Definir una función en Python que implemente DWSC

```
[ ] def separable_conv(input, ch, idx):
        x = layers.DepthwiseConv2D((3,3), padding="same", name='dwc0_dwsc'+str(idx))(input)
        x = layers.BatchNormalization( name='bn0_dwsc'+str(idx))(x)
        x = layers.Activation("relu", name='act0_dwsc'+str(idx))(x)
        x = layers.Conv2D(ch, (1,1), padding="same", name='conv0_dwsc'+str(idx))(x)
        x = layers.BatchNormalization(name='bn1_dwsc'+str(idx))(x)
        return layers.Activation("relu", name='act1_dwsc'+str(idx))(x)
```

4. Datasets

https://www.cs.toronto.edu/~kriz/cifar.html

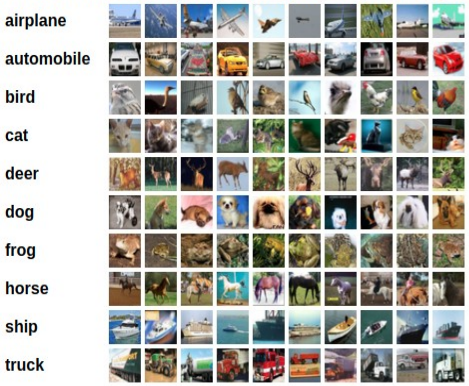
The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:



Descarga del dataset CIFAR-10

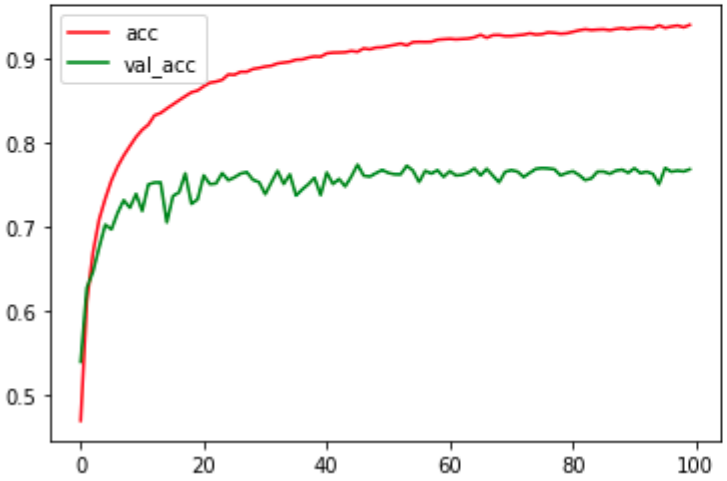
```
[ ] (train_imgs, train_lbls), (test_imgs, test_lbls) = datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 3s 0us/step
170508288/170498071 [==============================] - 3s 0us/step
```

5. Métricas

```
# Accuracy obtenida por epocas

plt.plot(history.history['accuracy'], label='acc', color='red')
plt.plot(history.history['val_accuracy'], label='val_acc', color='green')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7ff4cc1dbc90>
```



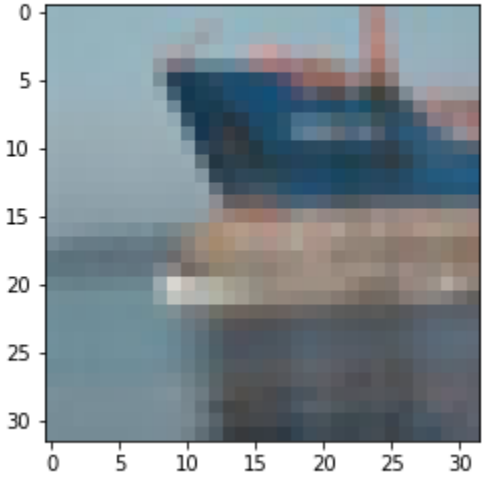
+ Código + Texto

```
[ ] print("Accuracy:", num_correct_samples/num_total_samples)
```

```
Accuracy: 0.7218
```

6. Resultados obtenidos

Etiqueta original es ship y etiqueta predicción ship




```
[227/227] Linking CXX executable zephyr/zephyr.elf
Memory region         Used Size  Region Size  %age Used
           FLASH:      137056 B       256 KB     52.28%
            SRAM:        8624 B        64 KB     13.16%
        IDT_LIST:           0 GB         2 KB      0.00%
v20@v20:~/zephyrproject/zephyr/samples/modules/tflite-micro/cifar10$ west build -t run
-- west build: running target run
[0/1] To exit from QEMU enter: 'CTRL+a, x'[QEMU] CPU: cortex-m3
qemu-system-arm: warning: nic stellaris_enet.0 has no peer
Timer with period zero, disabling
*** Booting Zephyr OS build zephyr-v3.0.0-2733-g13f382287ad0  ***
CORRECT classification! ship
qemu-system-arm: terminating on signal 2
ninja: build stopped: interrupted by user.
v20@v20:~/zephyrproject/zephyr/samples/modules/tflite-micro/cifar10$
```

7. Conclusiones

- **MovilNetV1 → Convolución Separable en profundidad.**
- **MovilNetV2 → ‘Linear Bottlenecks’ + ‘Inverted residual’.**
- **MovilNetV3 → Búsquedas complementarias, NAS y NetAdapt.**
- **MovilNetV3-Large.**
- **MovilNetV3-Small.**

8. Referencias

[1] Lin, J., Chen, W. M., Lin, Y., Gan, C., & Han, S. (2020). Mcunet: Tiny deep learning on iot devices. Advances in Neural Information Processing Systems, 33, 11711-11722.

[2] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[3] Lai, L., Suda, N., & Chandra, V. (2018). Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. arXiv preprint arXiv:1801.06601.(19/01/2018). https://arxiv.org/abs/1801.06601

[4] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture
search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 2820-2828).https://arxiv.org/abs/1807.11626

[5] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture
search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 2820-2828). Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 2820-2828).

[6] Mehta, S., Rastegari, M., Shapiro, L., & Hajishirzi, H. (2019). Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 9190-9200). https://arxiv.org/abs/1811.11431

[7] https://github.com/PacktPublishing/TinyML-Cookbook/blob/main/Chapter07/ColabNotebooks/prepare_model.ipynb

Gracias por su atención.