Reconocimiento facial y de voz en sistemas empotrados de bajo coste mediante el uso de TinyML

José Miguel Moreno[1], José Antonio de la Torre[1], Fernando Rincón[1], Julián Caba[1], José Luis Mira[1], Juan Carlos López[11]

Resumen— **Tradicionalmente los dispositivos embebidos han estado limitados a tareas de recolección y filtrado de datos, mientras que el procesamiento de estos datos se ha delegado en servidores con mayor capacidad de procesamiento en la nube. Sin embargo, con los últimos avances tecnológicos han surgido nuevos paradigmas como la computación en el Edge y en el Fog, acercando la capacidad de cómputo al dispositivo final. En este contexto, surge el concepto de TinyML, que busca implementar algoritmos de aprendizaje por computador directamente en el dispositivo final.**

En este trabajo, se evalúa la viabilidad del uso de técnicas avanzadas de aprendizaje por computador, como redes neuronales, en microcontroladores de bajo consumo. Específicamente, se ha implementado un sistema de reconocimiento facial y de voz para identificar a un usuario y gestionar su acceso a un área restringida. Se presenta la arquitectura de la solución implementada, logrando ejecutar ambos modelos dentro de dispositivo ARM Cortex M4 a 64 MHz, con una precisión del 84.4 % en el reconocimiento facial y del 90 % en el reconocimiento de voz.

Palabras clave— **TinyML, Deep Learning, Neural Networks, MobileNet, MFCC, Arduino, DSP, Low power**

I. Introducción

El término TinyML (Tiny Machine Learning) hace referencia a un concepto relativamente nuevo que involucra los sistemas embebidos y el aprendizaje por computador [1]. El TinyML traslada las técnicas utilizadas en aprendizaje por computador o *machine learning* en dispositivos de bajo consumo (en el orden de magnitud de milivatios) y de bajo coste. Normalmente, estos dispositivos se encuentran limitados en el uso de recursos, tanto a nivel de procesamiento como de memoria RAM y Flash.

Hasta ahora, el uso de machine learning en dispositivos embebidos se ha realizado a través de los despliegues IoT (Internet of Things) aprovechando las capacidades de comunicación de este tipo de dispositivos con el Cloud. Sin embargo, según el número de dispositivos en los despliegues ha incrementado, las técnicas tradicionales para la descarga de trabajo en servidores en el Cloud se han visto comprometidas, principalmente por la sobrecarga impuesta por la comunicación. Para solucionar estos inconvenientes se han propuesto otras técnicas como la computación en el Edge y en el Fog [2]. Este tipo de técnicas consiste en virtualizar las tareas realizadas típicamente en el Cloud y acercarlas a los dispositivos finales, reduciendo la latencia y mejorando la privacidad. Sin embargo, los dispositivos en el Edge, son dispositivos que normalmente están conectados a una fuente de energía y tienen altas capacidades de cómputo en comparación a los dispositivos embebidos finales.

El último paso en esta transición, resumida en la Figura 1, es ejecutar estos algoritmos en los dispositivos finales.

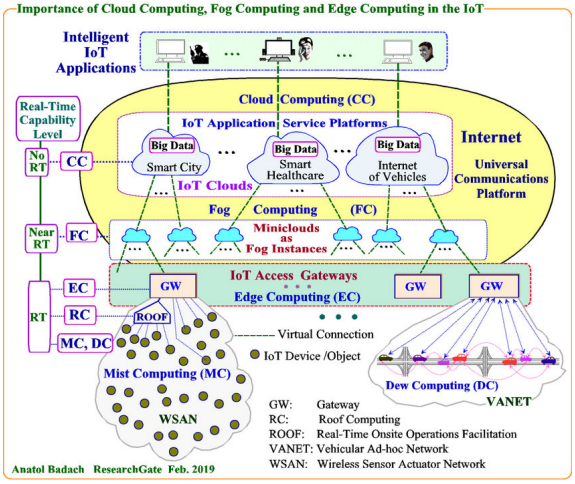

Fig. 1: Arquitectura Computing Continuum [3].

Tal y como se refleja en el estudio sistemático realizado por Hui Han y Julien Siebert [4], en el que se analizan los trabajos más relevantes en el área del TinyML, los casos de uso de TinyML van desde la detección de palabras clave hasta el reconocimiento de actividades deportivas (ver Figura 2). Uno de las primeras aplicaciones y, que más relevancia científica ha acaparado en los últimos años, es el de la detección de palabras claves. Muchos dispositivos inteligentes de grandes compañías como Google, Amazon o Microsoft utilizan este tipo de palabras para activar sus dispositivos y realizar las tareas para las que han sido diseñados. Tradicionalmente este tipo detección se ha realizado mediante una combinación de hardware especializado para la detección de palabras específicas y más tarde, una vez realizada la detección, el reconocimiento final en servidores en el Cloud. En los últimos años estas compañías han ido presentando nuevos modelos que permiten la inferencia total en local sin necesidad de servidores en el Cloud.

En este trabajo se presenta una implementación de un caso de uso para sistemas de seguridad mediante el uso de la voz y el reconocimiento facial. Se

[1]Dpto. de Tecnologías y Sistemas de Información, UCLM, 13071 Ciudad Real, España

pretende validar la viabilidad de este tipo de técnicas de TinyML para escenarios más complejos mediante la combinación de diversos modelos de aprendizaje dentro de un mismo dispositivo. El dispositivo embebido hace uso de las técnicas de TinyML para la detección inicial de personas mediante el reconocimiento facial para más tarde detectar e identificar una frase clave que permita la apertura de una puerta de seguridad. Finalmente, además de reconocer la frase se identifica a la persona usando la voz y su huella vocal.

II. Caso de uso

El objetivo principal del trabajo es validar la viabilidad de las técnicas de TinyML para el análisis directamente en el dispositivo final. El caso de uso plantea un reto debido al proesamiento de flujos de datos de imagen y audio dentro de un dispositivo embebido. En concreto se busca identificar tanto por voz como por imagen a una persona para su autorización de acceso a un área restringida. Este mismo caso de uso se podría usar en una cerradura inteligente.

El trabajo por tanto se ha dividido en tres 4 fases:

- Fase 1: Exploración del espacio de diseño: Se identifican las principales técnicas y herramientas para llevar a cabo el estudio.
- Fase 2: Implementación de la identificación facial
- Fase 3: Implementación de la identificación vocal
- Fase 4: Integración final: Despliegue en el microcontrolador

III. Desarrollo

A. Exploración del espacio de diseño

El campo del TinyML es un campo relativamente nuevo y las herramientas para el diseño e implementación todavía son escasas. En el trabajo [1] se realiza un análisis de las principales plataformas hardware y software utilizadas hasta la fecha. En esta revisión se concluye que la mayoría del hardware está basado en la arquitectura ARM. Esto se debe principalmente a las ventajas que ofrece este conjunto de instrucciones a la hora de realizar cómputo con un bajo coste energético [5]. En los últimos años ha aparecido un nuevo competidor a este conjunto de instrucciones que además de estas capacidades tiene la peculiaridad de ser de código abierto y sin coste por licencia, RISC-V. El uso de este nuevo conjunto de instrucciones todavía se encuentra enfocado en el entorno académico debido en parte a la falta de adopción por las compañías, aunque en los últimos años están apareciendo algunas como SiFive. En [6] se puede ver una implementación de un núcleo de bajo consumo basado en RISC-V aunque la implementación es experimental dentro de una FPGA (Field Programable Gate Array).

Basado en estos datos y la compatibilidad con el software que más adelante detallaremos, en este trabajo se ha usado la arquitectura ARM mediante el SoC (System on Chip) nRF52840 de la compañía Nordic. Este SoC está basado en el ARM Cortex-M4 y posee una gran cantidad de periféricos y sistemas de comunicación que lo hacen ideal para su uso en entornos IoT. Como placa de desarrollo hemos elegido el Arduino Nano 33 BLE Sense ya que incluye varios periféricos como el micrófono y el sensor de proximidad y está oficialmente soportado por la plataforma software que utilizaremos.

A la hora de elegir el hardware resulta indispensable tener en cuenta las tecnologías software que se van a usar. En la revisión realizada en [6] y [7] se realiza un estudio sobre las principales plataformas software centradas en TinyML. La principal librería para la implementación de modelos de aprendizaje en sistemas embebidos es Tensorflow Lite, desarrollada por Google y basada en su conocida librería Tensorflow. Esta librería se enfoca en el despliegue de redes neuronales en dispositivos de bajo cómputo principalmente mediante el uso de técnicas de post-procesado como la cuantización [8]. Este proceso permite reducir el uso de energía y el consumo de memoria. La ventaja principal de esta librería frente a otros es la compatibilidad prácticamente total con topologías de redes neuronales existentes únicamente teniendo que realizar el post-procesado final.

Sin embargo, el procesado de la red neuronal es solo uno de los pasos necesarios en las soluciones de machine learning. También es necesario el preprocesado de los datos, conexión con periféricos y comunicaciones. Para simplificar esta labor la empresa Edge Impulse proporciona una suite compatible con una gran variedad de dispositivos y librerías como Tensorflow Lite. En Edge Impulse los diferentes pasos dentro del flujo típico en los proyectos de TinyML se resumen en los mostrados en la Figura 3

- Adquisición de datos: Mediante un protocolo implementado en el módulo *edge-impulse-data-forwarder* se adquieren los datos directamente del dispositivo final o de cualquier dataset público mediante su interfaz web. La propia herramienta cuenta con un previsualizador puidiendo usar modelos preentrenados o mediante técnicas clásicas de reducción de la dimensionalidad como t-SNE ([9]) o PCA ([10]).
- Diseño del impulso: Dentro de Edge Impulse a la fase de procesamiento de datos e inferencia se le llama impulso. Los impulsos están divididos en 3 grandes bloques: bloque de entrada, bloque de procesamiento y bloque de aprendizaje. El bloque de entrada se encarga de adaptar los datos crudos en un conjunto de características adaptados a las necesidades concretas. Por ejemplo, en el caso de imágenes este bloque típicamente se encarga de hacer el redimensionado y de la imagen. El bloque de procesamiento, también llamado bloque DSP (Digital Signal Processing), es el encargado de generar las características de los datos mediante técnicas tradicionales, como la transformada de Fourier, cálculo de coeficientes de mel (MFCC), espectogramas, etc. Estos bloques son una parte importante de cualquier

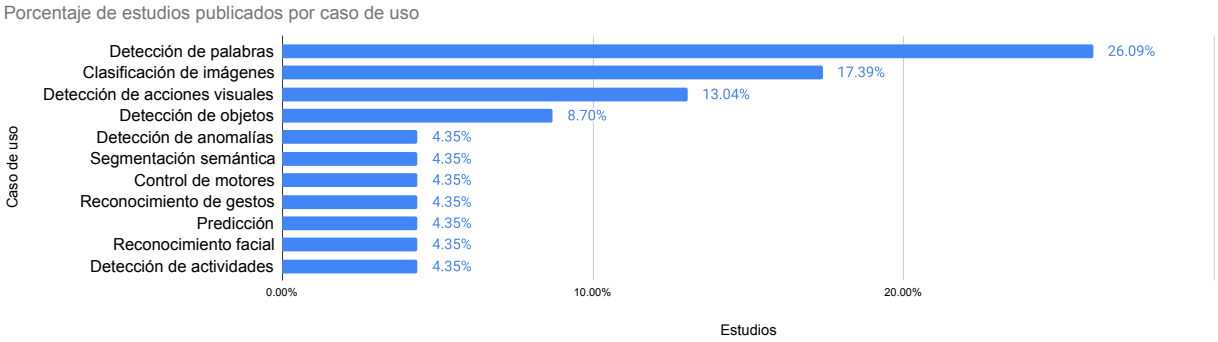

Fig. 2: Porcentaje de casos de uso más repetidos en las técnicas de TinyML [4].

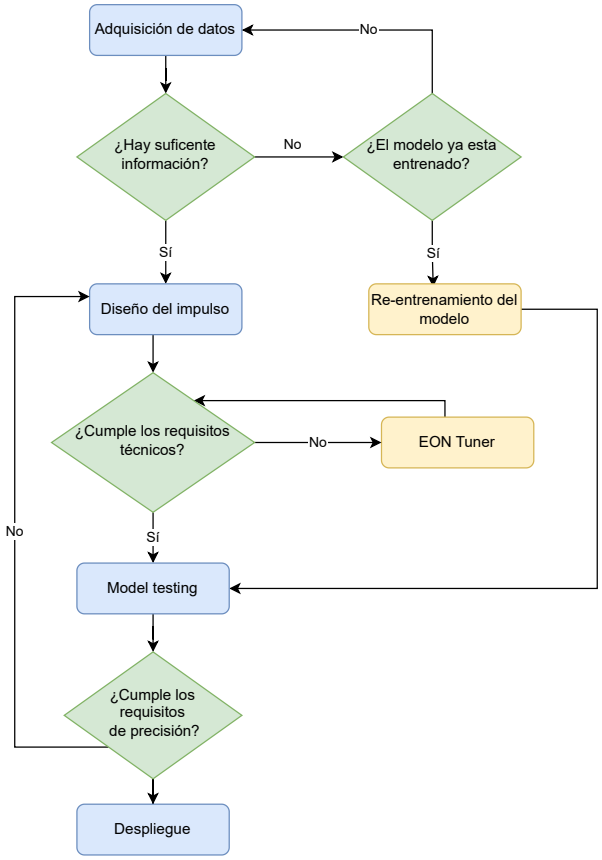

Fig. 3: Diagrama de flujo del proceso de desarrollo en Edge Impulse.

flujo de aprendizaje por computador y, por esta razón, Edge Impulse realiza una implementación adaptada a cada microcontrolador con el objetivo de utilizar las librerías y coprocesadores específicos de cada fabricante. Por último, el bloque de aprendizaje es el encargado de realizar la clasificación, detección, regresión, etc. en función del caso de uso a implementar. Edge Impulse da soporte para bloques de aprendizaje personalizados así como bloques con transferencia de conocimiento para reducir el tiempo de desarrollo.

- Post-Procesado: Tal y como se ha comentado la fase de post-procesado se encarga de cuantizar y adaptar los modelos para los dispositivos embebidos. La herramienta de Edge Impulse, EON Tuner analiza de forma automática los datos de entrada y el impulso diseñado para ofrecerte otras alternativas cambiando alguno de los bloques y cumpliendo con los requisitos de latencia y memoria impuestos por el usuario. Esta herramienta ayuda a reducir el tiempo de desarrollo mediante la reducción del espacio de diseño.
- Despliegue: En la fase de despliegue Edge Impulse proporciona varias alternativas en función del tipo de proyecto que se vaya a realizar. La opción más versátil es la encapsulación del proyecto en una librería C++ para su uso en firmware personalizado.

Debido a la gran versatilidad de la herramienta y su utilización de librerías establecidas dentro del área del machine learning en este proyecto se ha utilizado la herramienta Edge Impulse como software de desarrollo. Tal y como adelantamos, la placa Arduino Nano 33 BLE Sense esta soportada oficialmente por Edge Impulse por lo que se simplifica aun más el desarrollo mediante el uso de las capas HAL (Hardware Abstraction Layers) proporcionadas por Edge Impulse.

B. Implementación de la identificación facial

El reconocimiento facial de la persona autorizada se realiza mediante el pipeline representado en la Figura 4.

En primer lugar se realiza un preprocesado de la imagen a una resolución de 96x96. Con esta reducción conseguimos reducir los requisitos de memoria RAM y el consumo de energía en los bloques DSP posteriores. Además de este preprocesado se ha hecho uso del bloque *Image* que se encarga de normalizar la imagen y adaptar la profundidad de color para el bloque posterior. Este bloque se ha configurado para normalizar los valores RGB en un valor de punto flotante entre 0 y 1 para cada componente del espacio RGB. Esto permite centrar los valores de cada canal para que los bloques posteriores trabajen sobre el mismo espacio.

Tras la configuración del bloque DSP se añade una etapa de aprendizaje mediante el bloque de "transferencia de aprendizaje". La transferencia de aprendizaje permite solucionar problemas de clasificación, en este caso de imágenes, con un dataset pequeño [11]. En este caso Edge Impulse provee de varias redes preentrenadas para diferentes tamaños de imagen. Para

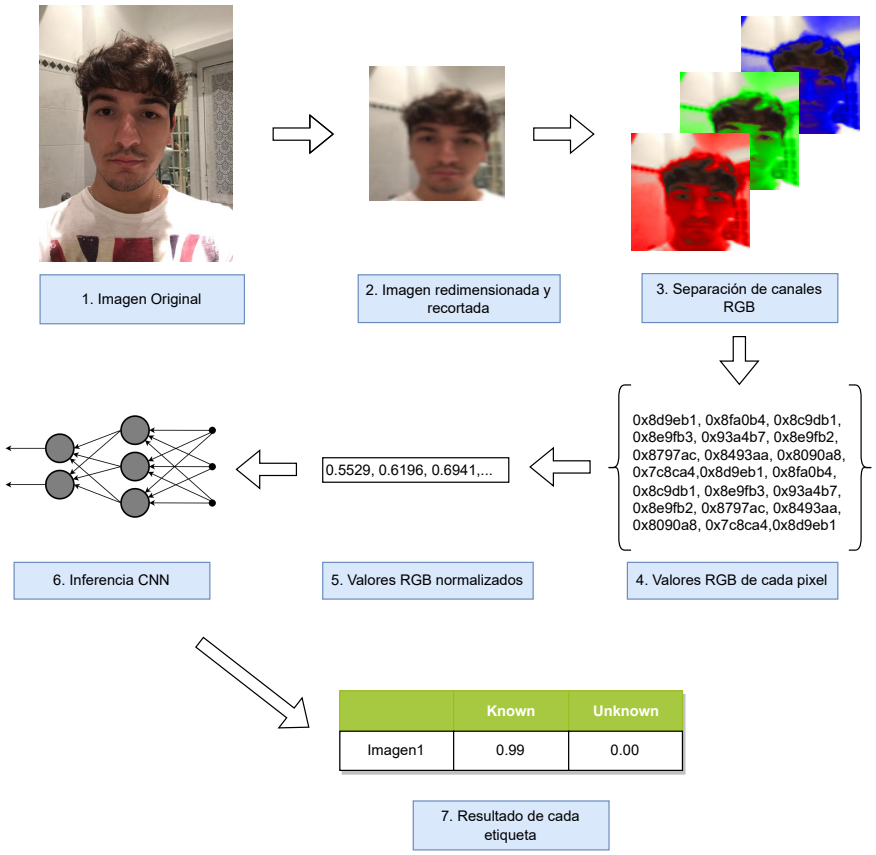

Fig. 4: Pipeline del reconocimiento facial con las diferentes etapas.

este trabajo se ha optado por MobileNetV1 [12]. Sobre esta red se ha añadido varias capas congelando el entrenamiento de las capas intermedias.

Para el entrenamiento se ha utilizado el dataset público de Kaggle [1] de imágenes de tipo *selfie* para las clases de personas desconocidas (no autorizadas). Por otra parte, para la clase de persona autorizada (autorizado) se han utilizado fotografías tomadas en el laboratorio. Finalmente, se han utilizado imágenes de diferentes fondos para proporcionar ejemplos de imágenes sin personas a la red. En la Tabla I se puede ver la matriz de confusión antes de la cuantización.

Finalmente, tras realizar el entrenamiento se ha cuantizado el modelo mediante el uso de enteros de 8 bits usando la herramienta Edge Impulse. Este paso permite reducir los requisitos de memoria y cómputo sin afectar de forma significativa a la precisión. La matriz de confusión tras la cuantización se puede ver en la Tabla II. Por otro lado, en la Tabla III se puede ver como ha variado tanto la precisión como el tiempo de procesamiento y el uso de memoria RAM antes y después de la cuantización. Como se puede observar el ahorro de recursos es sustancial aunque la precisión se ha visto afectada. Este tipo de compromiso entre cómputo y precisión es uno de los puntos fundamentales a la hora de evaluar la viabilidad en el despliegue de modelos de datos en dispositivos embebidos. Edge Impulse mediante EON Tuner simplifica mucho la tarea creando una matriz de configuraciones que resume el espacio de diseño aportando las diferentes estimaciones de métricas (Flash, RAM, CPU) después del postprocesado.

[1] `https://www.kaggle.com/datasets/tapakah68/selfies-id-images-dataset`

Tabla I: Matriz de confusión.

	Fondo	Autorizado	Desconocido
Fondo	100 %	0 %	0 %
Autorizado	0 %	80 %	20 %
Desconocido	3.3 %	5 %	91.7 %
F1	0.99	0.85	0.89

Tabla II: Matriz de confusión tras la cuantización a 8 bits.

	Fondo	Autorizado	Desconocido
Fondo	90.4 %	9.6 %	0 %
Autorizado	0 %	82.5 %	17.5 %
Desconocido	3.3 %	18.3 %	78.3 %
F1	0.94	0.73	0.82

C. Implementación de la identificación vocal

Para la identificación vocal se ha implementado el pipeline mostrado en la Figura 5. Como se puede ver, la estructura general es similar a la de la identificación facial, es decir, procesamiento de datos, DSP, aprendizaje y clasificación.

El audio de entrada está preparado para 16000 Hz y se aplica una ventana deslizante de 1 segundo con 1 segundo de salto. Estos valores se pueden adaptar en función del caso de uso y del sonido a detectar, en nuestro caso queremos detectar la palabra

Tabla III: Comparación del consumo de recursos antes y después de la cuantización en el modelo facial.

	Precisión	Tiempo	RAM	Flash
Original	92.5 %	7836 ms	243.3 KB	578.9 KB
Cuantizado	84.4 %	790 ms	100.3 KB	227.4 KB

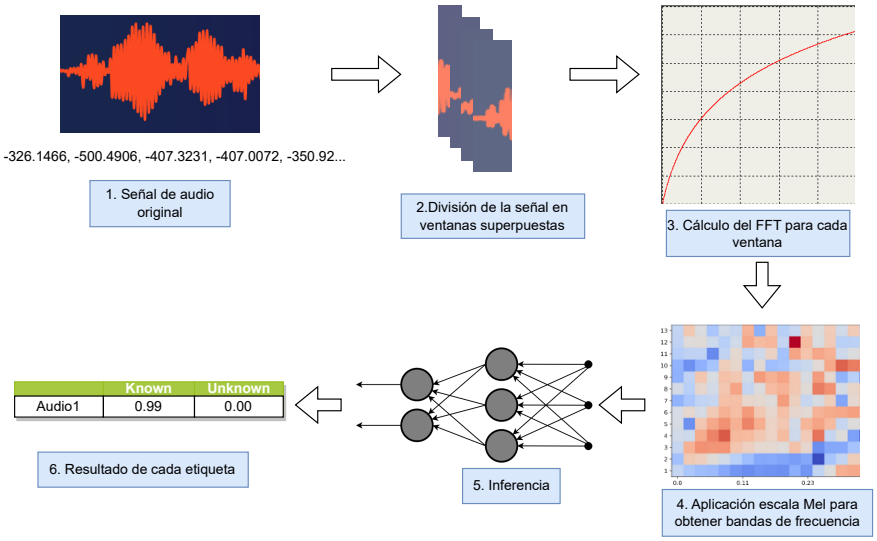

Fig. 5: Pipeline del reconocimiento de voz con las diferentes etapas.

«apertura», de un sujeto autorizado, de otras palabras, otros sujetos o sonido de fondo.

Analizar el audio de manera directa no resulta viable ya que se generan muchas características y el consumo de recursos sería alto sin una mejora en la precisión. Para mejora la clasificación hemos utilizado un bloque DSP llamado MFCC (Mel Frequency Cepstral Coefficients) que consiste en una serie de filtrados y procesamientos que reduce el número de características a clasificar. En la Tabla IV se pueden ver los coeficientes que hemos utilizado para el bloque. Estos coeficientes forman parte de los llamados hiperparámetros que se podrían validar mediante diferentes técnicas [13].

Tabla IV: Coeficientes "Mel Frequency Cepstral".

Coeficientes	13
Longitud	0.02
Stride	0.02
N Filtros	32
Longitud FFT	256
Tamaño ventana	101
Min Freq	0 Hz
Max Freq	8000 Hz
Pre Coef	0.98

Es importante tener en cuenta que en función del número de bloques DSP utilizados el consumo aumentará considerablemente. El bloque MFCC tiene un coste teórico de 294 ms de procesamiento y 13 KB de memoria RAM. La clasificación se ha realizado mediante una red neuronal de 6 capas. El número de entradas a la red es de 650 características que corresponden a los coeficientes calculados por el bloque DSP MFCC. En la Tabla V se puede ver la matriz de confusión una vez entrenado y cuantizado el modelo. Resulta interesante remarcar que el consumo de CPU de ejecutar la red neuronal es de apenas 5 ms y 3.7 KB de RAM en comparación a los 294 ms del bloque DSP. Es por esta razón que Edge Impulse utiliza las librerías del fabricante para el cálculo matemático con el objetivo de optimizar al máximo dichos bloques.

Para el entrenamiento de este pipeline se ha utilizado un dataset generado en el laboratorio con audios de hombres y mujeres. Cada sujeto ha grabado varios audios diciendo diferentes frases algunas con la palabra clave (apertura) y otras sin decir la palabra. Los datos se han dividido en un 78 % para entrenamiento y un 22 % para validación y se ha autobalanceado mediante la opción proporcionada por Edge Impulse. Tras la cuantización se obtiene una precisión del 90.8 %.

D. Despliegue

Una vez entrenados ambos modelos se han desplegado dentro del microcontrolador nRF52840 del Arduino Nano BLE 33 Sense. Para realizar el despliegue Edge Impulse proporciona varios métodos que se pueden dividir en dos grandes categorías.

La primera categoría son los despliegues autocontenidos que Edge Impulse proporciona para sus placas soportadas. En estos casos se proporciona un binario con todo el código listo para ser programado en la placa. Este código viene con un protocolo serie que te permite controlar el impulso y obtener los resultados del mismo. Este tipo de despliegues es útil para realizar medidas o para validar el diseño. Sin embargo, resulta imposible adaptar este firmware a un programa ya existente. Para soportar esta última opción Edge Impulse da la opción de exportar el pro-

Tabla V: Matriz de confusión de la clasificación de audio tras cuantización.

	Autorizado	Ruido	Otro hombre	Otra mujer	Desconocido
Autorizado	90 %	0 %	5 %	0 %	5 %
Ruido	0 %	100 %	0 %	0 %	0 %
Otro hombre	5.6 %	0	83.3 %	5.6 %	5.6 %
Otra mujer	0 %	10 %	0 %	80 %	10 %
Desconocido	0 %	0 %	9.1 %	0 %	90.9
F1	0.92	0.98	0.81	0.84	0.91

yecto como una librería C++ o Arduino en el caso que se desee utilizar dicho framework. En nuestro caso se ha utilizado la librería C++ por la versatilidad.

La librería se divide 3 componentes principales. El primero de los componentes es el SDK de edge-impulse que se mantiene constante entre los diferentes proyectos. Este SDK tiene las APIs de alto nivel y las adaptaciones a cada uno de los microcontroladores. El código es open-source y se puede acceder desde github[2]. Además por cada modelo se definen los metadatos del modelo y los pesos y operaciones de las redes neuronales y bloques DSP. Debido a una limitación a la hora de realizar la autogeneración de las librerías desde la plataforma no se pueden fusionar ambos impulsos en un mismo programa ya que se redefinen símbolos. Esta limitación está impuesta por Edge Impulse y, en caso de usar directamente Tensorflow Lite, evitaríamos este problema.

IV. Resultados

Una de las ventajas principales de las técnicas de TinyML es la reducción en el consumo de energía y reducción en la latencia. En este trabajo nos hemos centrado principalmente en reducir el consumo ya que la latencia no es relevante siempre que se mantenga dentro de un rango razonable.

Para la obtención de resultados y la comparación con las mismas implementaciones mediante soluciones tradicionales se han planteado 3 escenarios diferentes. El primer escenario pretende simular el cómputo en un procesador de alto rendimiento como sería el uso del Cloud mediante la arquitectura x86-64. En el segundo escenario se ha utilizado una placa SBC (Single Board Computer) con las capacidades típicas de un dispositivo en el Edge. Finalmente, el último escenario es el desarrollado en este trabajo que representa el cómputo en el dispositivo final.

Para sistematizar la recogida de resultados se ha alterado ligeramente el firmware para leer los datos de prueba (imágenes y fuentes de audio) desde el almacenamiento externo. También se ha añadido una rutina para monitorizar el tiempo de inferencia y procesamiento. En el caso de los escenarios basados en microprocesador, es decir, el Cloud y el Edge, se ha añadido un monitor que permite monitorizar el porcentaje de uso de CPU y RAM. Para sincronizar las medidas entre el monitor y el proceso de inferencia se hace uso de señales. El conjunto de pruebas consiste en la clasificación de 294 imágenes en diferentes resoluciones y 100 muestras de audio cada una de ellas de un segundo de duración. Tras realizar las pruebas múltiples veces se han obtenido los resultados que se pueden ver en la Tabla VI y en la Figura 6, Figura 7.

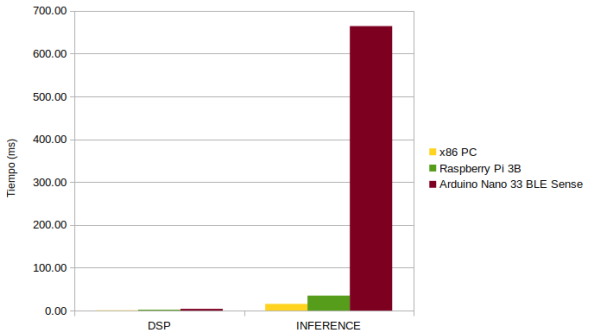

Fig. 6: Tiempo de inferencia y de procesamiento DSP en cada plataforma para el impulso de reconocimiento facial.

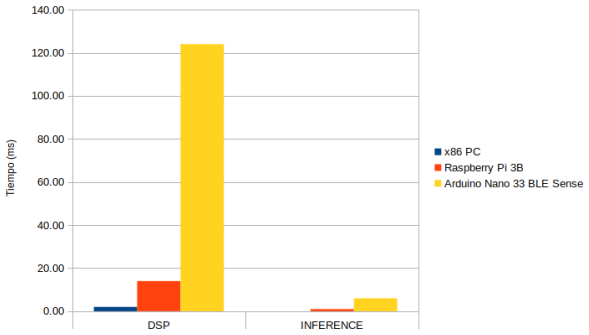

Fig. 7: Tiempo de inferencia y de procesamiento DSP en cada plataforma para el impulso de reconocimiento vocal.

Como se puede observar los tiempos tanto de DSP como sobretodo de inferencia son mucho mayores entre la implementación en x86 y en el microcontrolador. Sin embargo, el uso de RAM es mayor en el primer caso. Además se puede observar un uso de CPU del 90.01 % en el caso del x86 lo cual hace que el consumo energético sea considerablemente superior al caso del microcontrolador.

Por último, en este estudio no se ha tenido en cuenta el consumo de la transferencia de datos en comunicaciones que se debería realizar en el caso de delegar el procesamiento del microcontrolador al Cloud. Este cálculo, supone uno de los factores principales a la hora de valorar la descarga del cómputo entre los diferentes niveles Cloud, Edge, Fog [14]. En el caso de incluirlas se vería favorecido, aun más, el despliegue en el dispositivo final.

[2] `https://github.com/edgeimpulse/inferencing-sdk-cpp`

Tabla VI: Resultados de las pruebas sintéticas.

		x86	Raspberry Pi 4	Arduino
Imagen	DSP	0.67 ms	2.00 ms	4.00 ms
	Inferencia	15.67 ms	35.00 ms	664 ms
	CPU	90.01 %	100 %	100 %
	RAM	20.8 MB	16.02MB	100.3 KB
Audio	DSP	2.00 ms	14.00	124.00 ms
	Inferencia	0 ms	1.00 ms	6.00 ms
	CPU	2.13 %	6.82 %	100 %
	RAM	12.53 %	6.15 MB	13 KB

V. Conclusiones

En este trabajo se ha validado la efectividad de las técnicas de TinyML para desplegar algoritmos complejos de aprendizaje por computador en dispositivos de bajo coste. Se ha obtenido una precisión del 84.4 % en la tarea de reconocimiento facial y un 90 % en el reconocimiento de voz. Además se ha conseguido reducir el consumo de RAM y Flash lo suficiente para incluir ambos modelos en un mismo dispositivo. Se puede concluir por tanto que la tecnología está lo suficientemente madura para realizar modelos los suficientemente precisos dentro de un dispositivo embebido. La limitación principal de estas soluciones está en el uso de RAM y, aunque las técnicas de cuantización han mejorado lo suficiente para que la relación precision/energía sea cada vez mayor los dispositivos embebidos cuentan con una cantidad muy limitada de recursos. En base a los datos obtenidos y los resultados experimentales creemos que la combinación de estas técnicas con el cómputo cercano en el Edge pueden suponer la mejor alternativa. El primer filtrado se puede realizar directamente en el dispositivo final para enviar únicamente los casos en los que no se tenga la suficiente certeza a los dispositivos más cercanos en el Edge.

Agradecimientos

Esta investigación está parcialmente financiada por el Ministerio de Economía y Competitividad (MINECO) del Gobierno de España a través de los proyectos TALENT (PID2020-116417RB-C4, subproyectos 1 y 4) y MIRATAR (TED2021-132149BC41) y por el programa Europeo Horizonte 2020 bajo el proyecto SHAPES (GA Nº 857159).

Referencias

[1] Taiwo Samuel Ajani, Agbotiname Lucky Imoize, and Aderemi A. Atayero, "An overview of machine learning within embedded and mobile devices–optimizations and applications," *Sensors*, vol. 21, no. 13, 2021.

[2] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, New York, NY, USA, 2012, MCC '12, p. 13–16, Association for Computing Machinery.

[3] Anatol Badach, *IIoT – Intelligent IoT*, p. 23, WEKA Media, 03 2019.

[4] Hui Han and Julien Siebert, "TinyML: A Systematic Review and Synthesis of Existing Research," in *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Jeju Island, Korea, Republic of, Feb. 2022, pp. 269–274, IEEE.

[5] Teodor Neagoe, Ernest Karjala, and Logica Banica, "Why arm processors are the best choice for embedded low-power applications?," in *2010 IEEE 16th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 2010, pp. 253–258.

[6] Satyajit Bora and Roy Paily, "A high-performance core micro-architecture based on risc-v isa for low power applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 6, pp. 2132–2136, 2021.

[7] Muhammad Shafique, Theocharis Theocharides, Vijay Janapa Reddy, and Boris Murmann, "Tinyml: Current progress, research challenges, and future roadmap," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1303–1306.

[8] Ioan Lucan Orăşan, Ciprian Seiculescu, and Cătălin Daniel Caleanu, "Benchmarking tensorflow lite quantization algorithms for deep neural networks," in *2022 IEEE 16th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 2022, pp. 000221–000226.

[9] Laurens Van Der Maaten and Geoffrey Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579 – 2625, 2008, Cited by: 23201.

[10] Hervé Abdi and Lynne J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433 – 459, 2010, Cited by: 5880.

[11] Sinno Jialin Pan and Qiang Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345 – 1359, 2010, Cited by: 13539.

[12] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[13] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng, "Hyperparameter optimization for machine learning models based on bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26 – 40, 2019, Cited by: 470.

[14] Carlos Pereira, António Pinto, Duarte Ferreira, and Ana Aguiar, "Experimental characterization of mobile iot application latency," *IEEE Internet of Things Journal*, vol. 4, no. 4, pp. 1082 – 1094, 2017, Cited by: 30.