UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"Stato dell'arte del machine learning su microcontrollori"

Relatore: Prof. Bellotto Nicola

Laureando: Boldrin Luigi

ANNO ACCADEMICO 2022 – 2023

Data di laurea 21/03/2023

Indice

Capitolo 1

1. Introduzione

1.1 Cos'è e perché è importante un microcontrollore

Quotidianamente siamo abituati a sentir nominare il termine "processore", presente nei nostri smartphone, tablet o computer e siamo in grado di descriverne la funzione in modo più o meno specifico, anche nominandone il modello.
Il termine microcontrollore, invece, in quanto spesso utilizzato in contesti nei quali la potenza di calcolo non è una necessità primaria, non è generalmente una parola di cui facciamo uso.
Un microcontrollore è un processore, o meglio, un microprocessore, che include memoria e molti altri vari componenti nello stesso chip. Il risultato comporta una riduzione di prestazioni a favore però di una forte diminuzione delle dimensioni e del costo di produzione: i componenti che in un normale computer sono separati vengono uniti in un unico calcolatore.
Quando Intel presentò il suo primo processore nel 1971 vi era già una richiesta di microcontrollori da utilizzare su registratori di cassa, orologi e strumenti di misurazione, hanno quindi sempre avuto un ruolo importante nello sviluppo delle tecnologie che utilizziamo comunemente.
Nel 1974 venne presentato dall'americana Texas Instruments il TMS 1000, che includendo RAM, ROM e I/O direttamente sul chip, può essere considerato uno dei primi esempi di microcontrollore.
Al giorno d'oggi ne vengono prodotti miliardi all'anno e vengono utilizzati in ogni tipo di strumento tecnologico, dagli elettrodomestici alle automobili, dagli smartphones agli aerei.

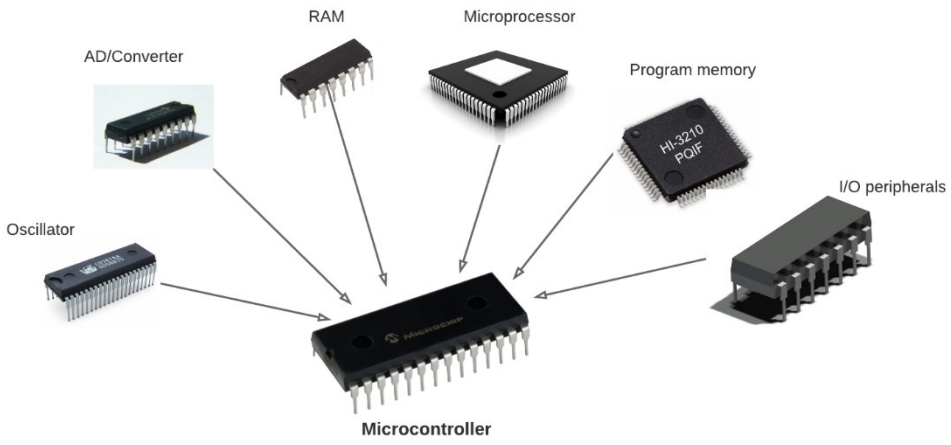

Figura1.1 Componenti di un microcontrollore

1.2 Cos'è e perché è importante il machine learning

Nonostante la rivoluzione che l'intelligenza artificiale sta portando sia un fenomeno relativamente recente, la sua nascita risale agli anni '50 del secolo precedente. Questo quindi porta a chiedersi per quale motivo i suoi effetti sembrino essere ristretti all'ultimo ventennio. La motivazione principale è la mancanza, in quel periodo, della potenza di calcolo necessaria ad ottenere risultati accurati in tempi accettabili su problemi che avessero un'utilità effettiva; per questo motivo l'intelligenza artificiale perse interesse e finanziamenti.
Grazie, però, agli enormi sviluppi nell'ambito della realizzazione di calcolatori sempre più potenti ed efficienti l'IA ha vissuto una crescita esponenziale negli ultimi tempi.
"Machine learning" è un termine che sentiamo sempre più spesso al giorno d'oggi, soprattutto in contesti legati a grandi quantità di dati e sono proprio questi "big-data" il secondo ingrediente fondamentale che permette ai computer di apprendere in maniera autonoma.
Il machine learning è quindi una tecnica di apprendimento computazionale, ovvero, il computer "studia" una serie di dati che gli vengono forniti e attraverso una lunga serie di calcoli, crea un modello che sia in grado di inferire delle informazioni su dati diversi da quelli forniti, ottenendo risultati accurati.
Al giorno d'oggi siamo circondati dagli effetti di questi algoritmi computazionali, dagli sviluppi in ambito medico ai veicoli a guida autonoma; in particolare due ultimi esempi sono divenuti molto famosi: DALL-E 2 e CHAT-GPT 3, che hanno fatto molto discutere sul futuro di queste tecnologie.
Che il machine learning sia eccitante e promettente è qualcosa che sembra scontato a molti, ma, nonostante ciò, c'è ancora molto lavoro da fare e ci sono molti margini di miglioramento.

1.3 Machine learning su microcontrollori

Nel 2014 un team di Google lavorava ad un progetto che permettesse il funzionamento di una rete neurale in un processore di segnale digitale (DPS), presente nella maggior parte degli smartphones. La rete in questione si occupava del riconoscimento della *wake-word* "OK Google", che permette di interfacciarsi in modo vocale con l'assistente. Tutto ciò, però, doveva essere realizzato entro un limite di spazio di circa 14 kilobytes, un numero minuscolo se confrontato con le decine di gigabytes occupati da una normale rete di deep learning. Questa forte limitazione di spazio deriva dal fatto che il DPS del dispositivo non possiede una grande quantità di memoria interna.

Questa prima necessità ci fa capire l'importanza che può avere l'utilizzo di reti neurali su processori alternativi, in grado di fornire buoni risultati con un grande risparmio di energia. Da qui nasce l'idea di definire TinyML il machine learning applicato a dispositivi che richiedano meno di 1mW di potenza, un numero che può sembrare arbitrario, ma in realtà permette al dispositivo di funzionare per un anno intero utilizzando solamente una semplice batteria a bottone. Questo vantaggio è ovviamente connesso ad un forte risparmio economico, ma ne conseguono altri aspetti positivi, quali una maggiore privacy visto il funzionamento autonomo della rete e una forte riduzione delle dimensioni.
Il mondo dell'intelligenza artificiale sta dunque "invadendo" strumenti e chip che un tempo nessuno avrebbe preso in considerazione per un utilizzo simile, nonostante ciò ci sono ancora molte sfide da superare.

Capitolo 2

2. Hardware, librerie e software utilizzati

2.1 Hardware

I microcontrollori, per via del loro utilizzo in applicazioni con necessità molto differenti, hanno caratteristiche tecniche spesso molto diverse tra loro. Vi sono quindi dispositivi in cui l'utilizzo di TinyML è stato testato ed ottimizzato, mentre per altri sono necessarie soluzioni alternative in attesa di software in grado di integrarsi in modo indipendente al microprocessore in dotazione.
A seguito ho selezionato alcune piattaforme hardware presenti all'interno del mercato attuale e compatibili con TinyML.

Hardware	Processor	CPU Clock	Flash	SRAM	Power/ Voltage	Connectivity	Sensors/Connectors	Organization
Apollo3	32-bit ARM Cortex-M4F	48 MHz, 96 MHz with TurboSPOT™	1 MB	384 KB	6µA/MHz Battery option,	BLE5, FTDI SPI, USB	Accelerometer, HM01B0 camera, MEMS microphone	SparkFun
STM32F Discovery	32-bit ARM Cortex-M4 FPU Core	48 MHz	1 MB	192 KB	3–5 V	LQFP100 I/O, USB	Accelerometer, microphone,	STMicroelectronics
ST IoTDiscovery	ARM Cortex-M4	48 MHz	1 MB,64Mbit Quad-SPI	128 KB	Battery option	8.211b/g/n, NFC, 868/915 MHz, BLE 4.1, USB	Microphone, accelerometer, gyroscope, barometer, gesture detection, humidity, temperature,	STMicroelectronics
ECM3532 AI Sensor NSP	ARM Cortex-M3, NXP CoolFlux 16-bit DSP	100 MHz	512 KB	256 KB	5µA/MHz, Battery option	BLE 4.2, RF, USB	Pressure, temperature, gyroscope, accelerometer, microphone	Eta Compute
Arduino Nano 33 BLE Sense	nRF52840	64 MHz	1 MB	256 KB	3.3 V, 15 mA/pin	UART, SPI, SPI, I2C, USB, BLE	IMU, microphone, gesture, light, proximity, barometer, temperature, humidity	Arduino
OpenMV Cam H7 Plus	ARM Cortex-M7	480 MHz	2 MB (Internal)	1 MB, 32 MB SDRAM	3.7 V Li-Ion	I2C, USB, CAN, UART,	5MP Camera at 50 FPS	OpenMV
Himax EW-I Plus	32-bit ARC EM9D DSP with FPU Core	400 MHz	2 MB	2 MB	1.2–3.3 V, Battery	SPI2, I2C, UART, USB	VGA Camera 60 FPS, accelerometer, microphone	SparkFun
Thunderboard Sense 2	EFR32™ Mighty Gecko Wireless SoC	38.4 MHz	1 KB	256 KB	3.3–5 V, Coin cell, ULP	2.4 GHz, USB, SPI,	Temperature, humidity, ambient light, pressure, air quality, microphone, hall-effect, UV	Silicon Labs
Sony's Spresense	ARM Cortex-M4F 6 Core	156 MHz	8 MB	1.5 MB	3.3–5 V	SPI, I2C, UART, I2S, GNSS antenna	Microphone, camera	Sony
TinyML Board	Syntiant® NDP101 NDP, 32-bit ARM Cortex-M0	48 MHz	256 KB	32 KB	3.7–5 V, LiPo battery	UART, I2C	Motion, microphone	Sytiant

Figura 2.1 Confronto tra piattaforme che supportano TinyML

La tabella mette a confronto le varie piattaforme osservando il tipo di processore utilizzato, la frequenza di clock della CPU, la quantità di memoria flash, la quantità di SRAM disponibile, la potenza o voltaggio richiesti, i tipi di connettività disponibili, i sensori presenti ed il produttore della medesima.

Osserviamo che la frequenza del processore è, in media, nell'ordine delle decine di MHz, la memoria flash è di circa 1 MB con meno di 1MB di SRAM. Questi valori sono considerevolmente inferiori rispetto alle specifiche di un moderno computer e ci aiutano a capire quanto sia importante utilizzare al meglio le risorse disponibili.

Notiamo, inoltre, che i dispositivi sono muniti di molti sensori, tra i quali: accelerometro, sensore di temperatura e umidità, giroscopio, microfono, sensore di prossimità, riconoscitore di gesti e sensore fotografico.

Il consumo medio è di pochi milliwatt e molti dispositivi possono essere alimentati a batteria oltre che dal comune collegamento a filo.

Si nota, inoltre, che l'ARM Cortex-M4 è il processore più comune tra le alternative proposte per via del suo ottimo compromesso tra costi, efficienza e consumi.

Nonostante l'uso di microprocessori già esistenti garantisca una flessibilità maggiore, architetture specializzate possono arrivare ad ottenere performance di circa un micro Joule per inferenza, andando ad ampliare le possibilità future di queste tecnologie.

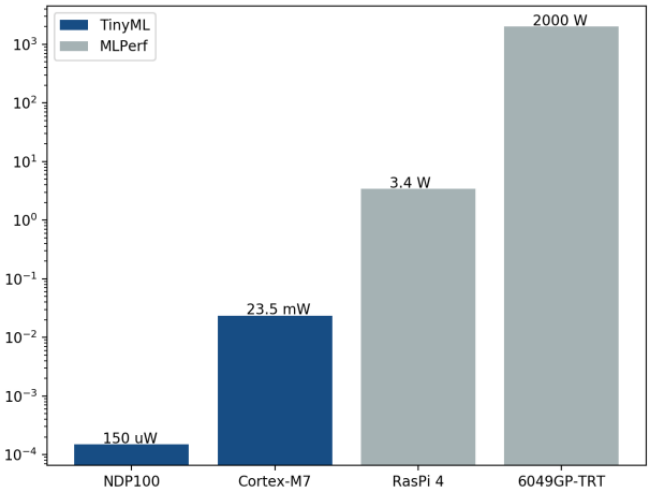

Figura 2.2 Confronto dei consumi energetici tra piattaforme differenti

2.2 Librerie e software

Tutte le possibilità qui elencate fanno parte di progetti in forte sviluppo e quindi, in quanto tali, non garantiscono risultati perfettamente privi di errori ed hanno forti margini di miglioramento.

- *TensorFlow Lite* (TFL) è un framework open-source per l'addestramento, la conversione, l'ottimizzazione e la distribuzione di modelli neurali su dispositivi mobili; in particolare, *TensorFlow Lite Micro* (TFLM) si occupa dei microcontrollori. TFLM è scritto in C++ 11, ha un *core runtime* che pesa solamente 16 KB e richiede una piattaforma a 32 bit per poter funzionare. Al momento, però, supporta solamente 12 piattaforme hardware ed un sottoinsieme limitato di operazioni TensorFlow, inoltre non è ancora possibile addestrare una rete sul dispositivo stesso.
  La rete deve essere addestrata su un dispositivo esterno utilizzando TensorFlow, viene poi salvata e quantizzata ed il file (in formato .tflite) viene successivamente convertito in un array di byte che verrà letto dal codice del microcontrollore.

- *uTensor* è framework basato su TFLM che però è stato ulteriormente alleggerito arrivando ad avere un core runtime di soli 2 KB. Contiene inoltre uno strumento per l'elaborazione grafica ed in futuro avrà un sistema per la collezione di dati.

- *Embedded Learning Library* (ELL) è una libreria open-source creata da Microsoft per il machine learning su dispositivi embedded quali Arduino e *micro:bit*. La libreria contiene un compilatore che converte un modello creato esternamente in codice binario che verrà letto dal dispositivo scelto.

- *NanoEdge AI Studio* è un software desktop user-friendly per lo sviluppo e la distribuzione di reti neurali su microcontrollori di tipo STM32. Il programma crea una libreria che si adatta alle richieste di progettazione per ottenere un risultato ottimale in termini di accuratezza e memoria utilizzata.

- *Edge Impulse* è un servizio su cloud per lo sviluppo di modelli di machine learning per dispositivi edge. L'addestramento viene eseguito online ed in seguito il modello può essere scaricato ed esportato sulla piattaforma desiderata.

- *Artificial Intelligence for Embedded Systems* (AIfES) è una libreria che permette l'utilizzo di reti neurali su un vasto numero di microprocessori, con architetture che vanno dagli 8 ai 64 bit. AIfES, contrariamente ad altre soluzioni, permette l'addestramento della rete sul dispositivo stesso. Unica limitazione è la sola possibilità di utilizzare reti neurali *feed-forward* (FNN), ma in futuro è prevista la possibilità di implementare anche reti convoluzionali (CNN).

- *µTVM* è l'estensione di un framework di machine learning (*Tensor Virtual Machines*) che mira a facilitare ed ottimizzare la distribuzione di reti neurali su microcontrollori. Al momento è ancora in forte fase di sviluppo e non supporta un gran numero di dispositivi.

- *Seedot* è un linguaggio domain-specific per la creazione di algoritmi di machine learning per microcontrollori. Introduce tecniche per la riduzione dello spazio di ricerca di parametri chiave e implementazioni efficienti di operazioni computazionalmente costose.

- *hls4ml* è un pacchetto di Python per la conversione di un modello preesistente in linguaggio HLS (*High Level Synthesis*) che permettere di configurarlo in base alle esigenze. Il progetto è ancora in fase di sviluppo e al momento non supporta una vasta tipologia di reti.

Capitolo 3

3. Algoritmi e metodi per il machine learning

3.1 Modelli

La maggior parte dei modelli utilizza reti neurali (NN) per il machine learning, ma non sono rare applicazioni basate su soluzioni non-NN quali *decision tree* e *Support Vector Machine*, nei casi in cui siano richiesti una potenza di calcolo ed un'occupazione di memoria minori. Vista la nascita recente del machine learning su dispositivi di tipo MCU (*Micro Controller Unit*), non ci sono dei modelli riconosciuti in modo diffuso come rappresentativi ed ottimali per un certo tipo di problema. Questo, fortunatamente, incoraggia la ricerca e lo sviluppo di nuove soluzioni che possano performare meglio di quelle già esistenti, facendo crescere la community di interessati.

Input Type	Use Cases	Model Types	Datasets
Audio	Audio Wake Words Context Recognition Control Words Keyword Detection	DNN CNN RNN LSTM	Speech Commands (Warden, 2018a) AudioSet (Gemmeke et al., 2017) ExtraSensory (Vaizman et al., 2017)
Image	Visual Wake Words Object Detection Image Classification Gesture Recognition Object Counting Text Recognition	DNN CNN SVM Decision Trees KNN Linear	Visual Wake Words (Chowdhery et al., 2019) CIFAR10 (Krizhevsky et al., 2009b) MNIST (LeCun & Cortes, 2010) ImageNet (Deng et al., 2009) DVS128 Gesture (Amir et al., 2017)
Physiological / Behavioral Metrics	Segmentation Forecasting Activity Detection	DNN Decision Tree SVM Linear	Physionet (Goldberger et al., 2000) HAR (Cramariuc, 2019) DSA (Altun et al., 2010) Opportunity (Roggen et al., 2010) UCI EMG (Lobov et al., 2018)
Industry Telemetry	Sensing (light, temp, etc) Anomaly Detection Motor Control Predictive Maintenance	DNN Decision Tree SVM Linear Naive Bayes	UCI Air Quality (De Vito et al., 2008) UCI Gas (Vergara et al., 2012) NASA's PCoE (Saxena & Goebel, 2008)

Figura 3.1 Sondaggio sui casi d'uso, modelli e dataset di reti TinyML

3.2 Calcolo iperdimensionale

Il calcolo iperdimensionale (HDC) è un'architettura computazionale ispirata al funzionamento dei circuiti neuronali all'interno del nostro cervello.

Questo paradigma permette di ottenere risultati in tempi più ristretti e con consumi di energia minori su problemi riconducibili ad applicazioni monodimensionali. Invece su applicazioni di tipo bidimensionale le CNN hanno un'accuratezza maggiore, ma anche un costo computazionale superiore.

Tutti i dati del problema vengono convertiti in vettori iperdimensionali ($D \geq 1000$) e mappati in uno spazio in cui vengono sommati linearmente ad altri vettori per ottenere un *ensable* di *class encoders*. Una volta completato il processo di addestramento il vettore di input chiamato *query hyper vector* viene confrontato, utilizzando la distanza di Hamming o la similarità del coseno, ai vari vettori rappresentativi delle rispettive classi, il class encoder più vicino sarà quello che deciderà il risultato della ricerca.

È possibile, inoltre, ridurre i tempi di ricerca quantizzando i vettori su uno spazio con meno dimensioni a discapito dell'accuratezza dei risultati.

3.3 Swapping

La scarsa quantità di memoria SRAM presente nei microcontrollori costituisce un forte limite al tipo di modelli neurali che è possibile utilizzare, senza dover sacrificare in accuratezza.

Una possibile soluzione a questo problema è l'utilizzo di una tecnica di swapping dei dati tra SRAM e memoria flash esterna. Questo metodo sembrerebbe poter introdurre problemi quali il rallentamento dovuto alle operazioni di I/O e l'usura eccessiva della memoria esterna.

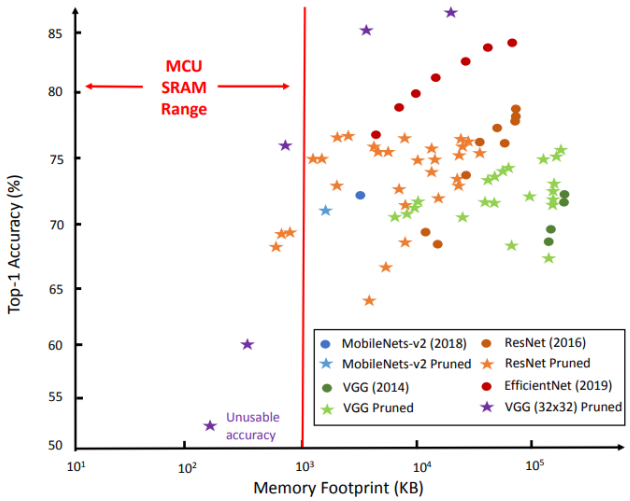

Figura 3.2 Confronto tra dimensioni e accuratezza di modelli neurali comunemente utilizzati

Si nota, però, che la prima complicazione riguarda solamente quelle operazioni che richiedono calcoli aritmetici meno complessi, invece per quanto riguarda operazioni come la convoluzione, il tempo di lettura/scrittura è minore di quello computazionale, evitando rallentamenti. L'usura della memoria, invece, ha un impatto minore di quanto si possa pensare: le operazioni sono per la maggior parte di lettura (che non danneggia la memoria) ed in generale non sono così frequenti. La figura 3.3 presenta l'architettura *SwapNN*, che analizzando la memoria disponibile, calcola le dimensioni dei segmenti in cui deve essere suddivisa la rete per permettere un'esecuzione fluida dell'algoritmo inferenziale.
Questo metodo sembra dare risultati molto promettenti, in particolare, la possibilità di utilizzare reti più complesse e modelli preesistenti senza perdita di precisione è un'importantissima aggiunta nell'ecosistema del TinyML.

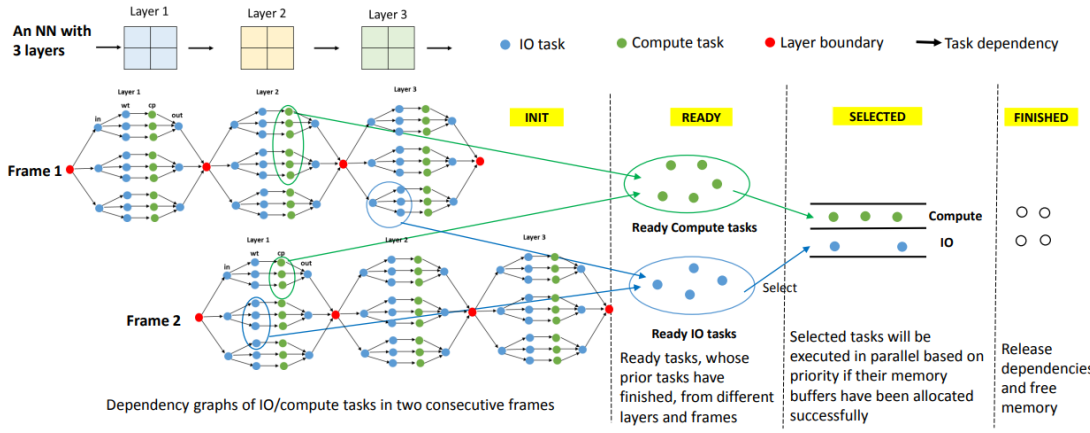

Figura 3.3 Presentazione dell'architettura SwapNN

3.4 Attention condensers

Gli *attention condensers* sono delle architetture stand-alone che permettono il perfezionamento di un modello di deep learning attraverso meccanismi di *self-attention.* Questi studiano i dati della rete e condensano selettivamente quelle parti che mostrano un livello di attivazione maggiore, in seguito queste zone assumono un importanza maggiore all'interno della rete, permettendo risultati di classificazione migliori.
Le possibilità di utilizzo di questo metodo sono varie, perciò è possibile studiare la soluzione migliore per il tipo di rete desiderato. Vediamo, infatti, nella tabella seguente, come diverse varianti dello stesso modello abbiano vantaggi diversi tra loro.

Model	Test Accuracy	Params	Mult-Adds
trad-fpool13 [11]	90.5%	1370K	125M
tpool2 [11]	91.7%	1090K	103M
TDNN [14]	94.2%	251K	25.1M
res15-narrow [12]	94.0%	42.6K	160M
PONAS-kws2 [15]	94.3%	131K	168M
TinySpeech-X	**94.6%**	10.8K	10.9M
TinySpeech-Y	93.6%	6.1K	6.5M
TinySpeech-Z	92.4%	**2.7K**	**2.6M**
TinySpeech-M	91.9%	4.7K	4.4M

Figura 3.4 Confronto tra diverse implementazioni di un algoritmo di self-attention

Questa architettura mostra risultati molto promettenti, ma per il momento non è stata testata in modo estensivo su altri tipi di problematiche quali il riconoscimento di oggetti o l'elaborazione di linguaggio naturale.

3.5 Ricerca dell'architettura neurale

Nel mondo del machine learning la scelta del modello e degli iperparametri di una rete è un passo fondamentale del percorso di addestramento. La differenza che queste scelte possono fare si ripercuote in modo significativo sui risultati finali della rete, per questo è molto importante studiare e testare tutte le varie possibilità. Lo spazio di ricerca è però molto vasto, tanto che vengono usati complicati algoritmi (*Neural Architecture Search*) per aiutare a trovare la soluzione migliore in base alle necessità progettuali.
Le reti NAS esistenti però si occupano di ricerca di modelli per utilizzo su dispositivi diversi dai microcontrollori, è quindi necessario sviluppare delle versioni alternative.
Un esempio è *µNAS*, creata appositamente per la ricerca, su uno spazio altamente granulare, della rete TinyML che risponda al meglio alle caratteristiche desiderate.
Nella pagina seguente è presentata una tabella che mette a confronto questo metodo con altri tipicamente utilizzati.

Dataset	Model	Acc. (%)	Model size	RAM usage	MACs	Difference
MNIST	SpArSe (Fedorov et al., 2019)	98.64	2770	$\geq$ 1960 B	*unk.*	Size $\uparrow$ 5.7×
	SpArSe	96.49	1440	$\geq$ 1330 B	*unk.*	Acc. $\downarrow$ 2.7%
	BonsaiOpt (Kumar et al., 2017)	94.38	490	$<$ 2000 B	*unk.*	Acc. $\downarrow$ 4.8%
	ProtoNN (Gupta et al., 2017)	95.88	63'900	$<$ 64'000 B	*unk.*	Acc. $\downarrow$ 3.3%
	µNAS (1174 steps, 1 GPU-day)	**99.19**	**480**	**488 B**	28.6 K	
CIFAR-10 (binary)	SpArSe	73.84	780	$\geq$ 1280 B	*unk.*	Acc. $\downarrow$ 3.7%
	µNAS (1206 steps, 2 GPU-days)	**77.49**	**685**	**909 B**	41.2 K	
Chars74K (binary)	SpArSe	77.78	460	$\geq$ **720 B**	*unk.*	Acc. $\downarrow$ 3.4%
	µNAS (743 steps, 0.5 GPU-day)	**81.20**	**390**	867 B	107 K	
Speech Commands	RENA (Zhou et al., 2018)	94.04	47 K	*unk.*	$\approx$700 M	MACs $\uparrow$ 636×
	RENA	**94.82**	67 K	*unk.*	$\approx$3'265 M	MACs $\uparrow$ 2968×
	DS-CNN (Zhang et al., 2017)	94.45	$<$**38.6 K**	*unk.*	$\approx$2.7 M	MACs $\uparrow$ 2.2×
	MCUNet (Lin et al., 2020a)	$\approx$91.20	$<$ 1 M	80 KB	*unk.*	Acc. $\downarrow$ 4.4%
	MCUNet	$\approx$**95.91**	$<$ 1 M	311 KB	*unk.*	RAM $\uparrow$ 14.7×
	µNAS (1960 steps, 39 GPU-days)	**95.58**	**37 K**	**21.1 KB**	**1.1 M**	
	MN-KWS (L) (Banbury et al., 2020)	**95.3**	612 K	208 K	*unk.*	Size $\uparrow$ 31.8×
	µNAS (1514 steps, 30 GPU-days)	**95.36**	**19.2 K**	25.7 KB	1.1 M	
Fashion MNIST	reported (Zalando, 2020)	92.50	$\approx$100 K	*unk.*	*unk.*	Size $\uparrow$ 1.6×
	µNAS (1161 steps, 3 GPU-days)	**93.22**	**63.6 K**	12.6 KB	4.4 M	
CIFAR-10	LEMONADE (Elsken et al., 2018)	$\approx$**91.77**	**10 K**	*unk.*	*unk.*	
	µNAS (4205 steps, 23 GPU-days)	86.49	11.4 K	15.4 KB	384 K	Acc. $\downarrow$ 5%
Chars74K	*µNAS (1755 steps, 2 GPU-days)*	82.65	3.85 K	9.75 KB	279 K	
	µNAS (1724 steps, 2 GPU-days)	76.05	13.9 K	1.81 KB	111 K	

Figura 3.5 Confronto di prestazioni tra modelli che utilizzano µNAS e altri modelli tipicamente usati

3.6 Compressione dei dati

La compressione dei dati è un aspetto fondamentale nel mondo della gestione informatica delle informazioni. Grazie ad essa è possibile immagazzinare e trasmettere ogni tipo di file utilizzando meno risorse, con risparmi di tipo economico e logistico.

Come abbiamo potuto osservare precedentemente i microcontrollori sono dispositivi dotati di una scarsa quantità di memoria e perciò è imperativo utilizzare efficaci soluzioni di compressione. Le scelte algoritmiche possibili sono varie: a seconda delle necessità progettuali è possibile privilegiare la velocità di decompressione, lo spazio occupato o la quantità di informazioni perse.

Il primo approccio preso in considerazione si chiama *Tiny Anomaly Compressor* (TAC) ed è basato sull'eccentricità dei dati, ovvero la loro deviazione rispetto allo standard.

Un vantaggio di questo metodo è che non richiede nessuna informazione riguardo la distribuzione matematica dei dati: l'algoritmo impara attraverso lo studio delle loro relazioni spaziali e temporali.

Questo metodo è stato confrontato con due algoritmi già esistenti (*Swing Door Trending* (SDT) e *Discrete Cosine Transform* (DCT)) e i risultati sono un errore di compressione ed un rapporto segnale/rumore migliori a parità di dimensione dei dati compressi.

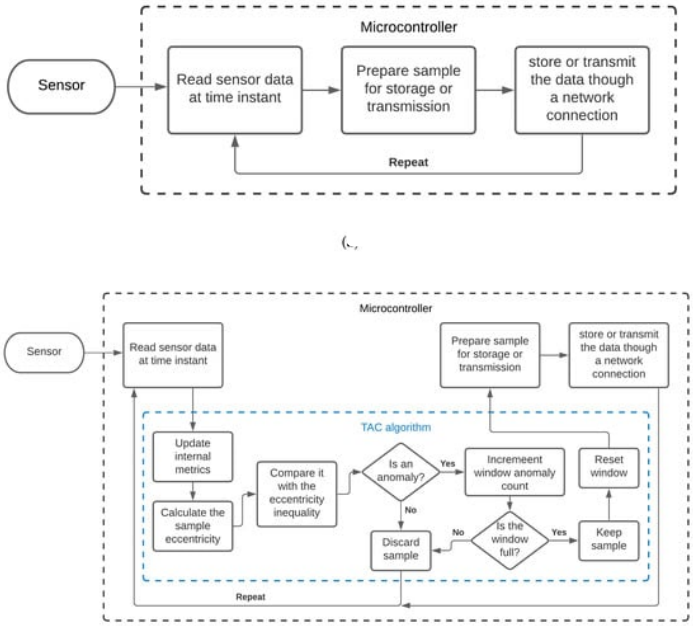

Figura 3.6 Schema del funzionamento dell'algoritmo TAC

Il secondo approccio utilizza i *prodotti di Kronecker* (KP), basati su operazioni matriciali, per comprimere reti RNN per dispositivi IoT, arrivando ad ottenere un fattore di compressione che può andare da 15 a 38 volte. Questo metodo però, se applicato a grandi reti di elaborazione del linguaggio (NPL), causa un diminuzione dell'accuratezza pari al 26%.
È stato quindi elaborato un algoritmo che possa migliorare questo risultato, attraverso l'aggiunta di uno schema di regolarizzazione chiamato *co-matrix dropout regularization* (CMR). Questo metodo di compressione è stato chiamato *doped kronecker product compression* e permette di ridurre un modello di rete LSTM da 25 MB 25 volte, con una perdita del *perplexity score* del 1.4%, contro il 15% ed il 27% di metodi alternativi.

Comparisons with prior art	Compression Factor	Test Perp
Baseline LM	1×	82.04
4-bit quant (Park et al., 2017)	8×	83.84
3-bit quant (Lee & Kim, 2018)	10.67×	83.14
Tensor Train (Grachev et al., 2019)	1.67×	168.639
Weight Distortion (Lee et al., 2018)	10×	84.64
Weight Distortion (Lee et al., 2018)	20×	93.39
DKP (Ours)	25×	83.24

Figura 3.7 Confronto tra DKP ed altri metodi di compressione

L'ultimo approccio considerato si occupa della compressione delle immagini, un aspetto fondamentale per tutte le applicazioni che utilizzino computer vision.
Il framework di compressione, denominato *starfish*, è stato progettato in particolare per dispositivi LWAN (*low-power wide area network*), dispositivi IoT che comunicano via wireless. L'algoritmo che si occupa della compressione è una vera e propria rete neurale di deep learning (DNN), che si configura in modo automatico a seconda delle architetture e degli hardware in uso. Il framework mostra risultati molto promettenti, ottenendo livelli di compressione fino a 3-4 volte maggiori rispetto ad un normale soluzione JPEG, tutto questo con un consumo di tempo ed energia fino a 2.5 volte minore.

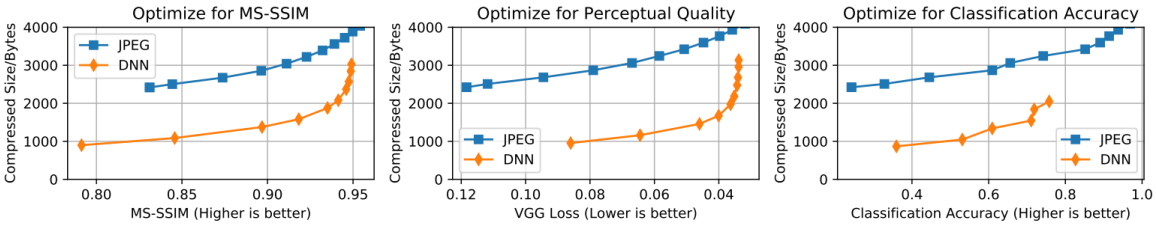

Figura 3.8 Confronto tra Starfish e JPEG con differenti ottimizzazioni

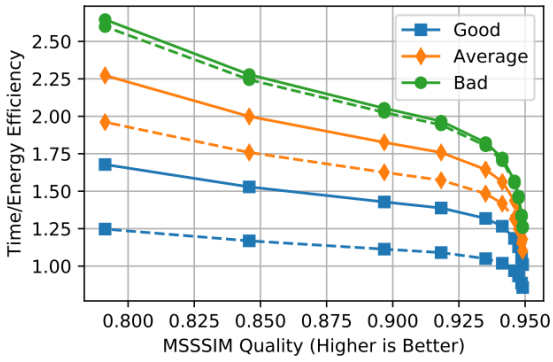

Figura 3.9 Confronto tra efficienza temporale ed energetica di Starfish con JPEG (2.00 significa due volte più efficiente), con condizioni di link differenti

3.7 Quantizzazione del modello

La quantizzazione è una tecnica di riduzione delle dimensioni di una variabile, tipicamente da 32 a 8 bit, che garantisce un risparmio di spazio in memoria ed un miglioramento nella velocità di esecuzione delle operazioni tra i dati.
Anche per questa tecnica gli esempi di applicazione sono molti, ne analizzo due in particolare. Il primo è un algoritmo di *post-training quantization* (PTQ), ovvero un algoritmo di quantizzazione che viene applicato in seguito all'addestramento della rete. Il processo, a differenza di un algoritmo di tipo *quantization-aware training* (QAT) è quindi libero di agire senza avere informazioni sulla pipeline della rete e non richiede un grande numero di dati di calibrazione. In particolare, l'algoritmo PTQ si occupa di quantizzazione a "bassa precisione", arrivando a comprimere le variabili originali in soli 2 bit.
I risultati mostrati da questo metodo sono promettenti ed è importante che vengano riconosciuti, in modo tale da poter essere aggiunti a librerie già esistenti quali TensorFlow lite o PyTorch Mobile.

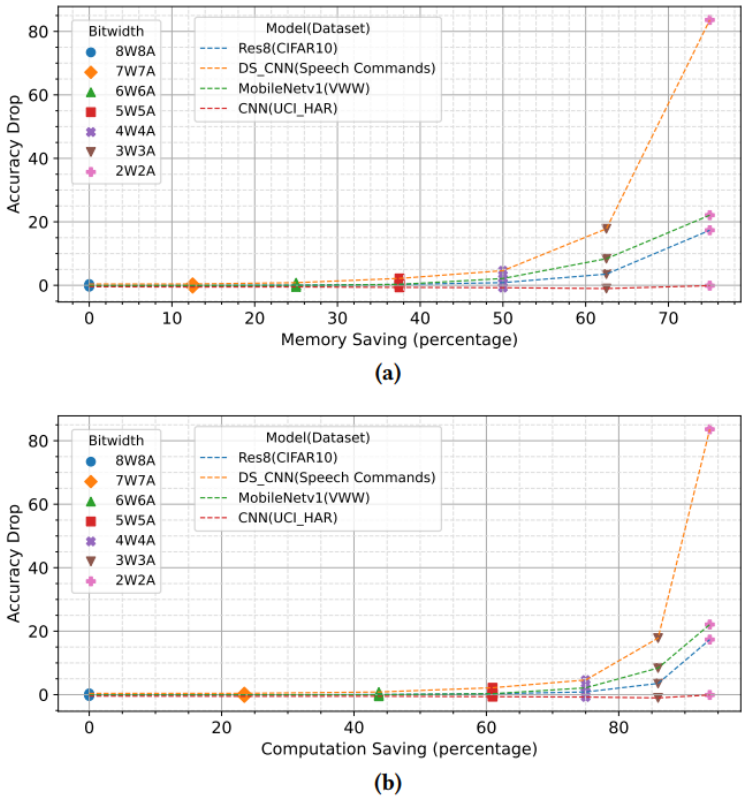

Figura 3.10 Confronto tra accuratezza e risparmio in memoria (a) / computazione (b) di vari modelli quantizzati. Notiamo che 4W4A (4 bit weights 4 bit activations) costituisce un buon compromesso.

Il secondo metodo, a differenza del primo, interviene direttamente sull'addestramento della rete ottenendo risultati migliori a parità di compressione.
Anche questo algoritmo può scendere sotto gli 8 bit di precisione e, a differenza di QAT, funziona influenzando direttamente la *loss function* del modello, penalizzando quei risultati che non verrebbero quantizzati efficacemente.

Model	Accuracy (%)	Δ Accuracy (%)	Size (KB)	Δ Size Reduction	Method, Precision
MobileNet V1 ($\alpha = 0.25$)	88.7	-	834	-	FP32
	72.2	-16.5	123	6x	PTQ, 4bit
	52.7	-36.0	73	11.4x	PTQ, 2bit
	87.9	-0.8	123	6x	QGT, 4bit
	82.0	-6.7	73	11.4x	QGT, 2bit
MobileNet V2 ($\alpha = 0.25$)	90.4	-	1440	-	FP32
	89.1	-1.3	133	10.8x	PTQ, 4bit
	68.4	-22.0	81	17.7x	PTQ, 2bit
	90.3	-0.1	133	10.8x	QGT, 4bit
	87.3	-3.1	81	17.7x	QGT, 2bit

Figura 3.11 Risultati dopo la quantizzazione di un modello di person detection. Si nota che QCT risulta migliore di PTQ.

Altri possibili esempi di quantizzazione sono TENT (*Tapered FixEd PoiNT*), *CMix-NN* e SWIS (*Shared Weight bIt Sparsity*).
Il primo sfrutta un metodo alternativo di memorizzazione delle variabili, il secondo è una libreria per la creazione di modelli già quantizzati su microcontrollori, il terzo invece è un framework di quantizzazione basato sulla decomposizione dei pesi e su algoritmi di scheduling.

3.8 OFA Networks

Un OFA (*one-for-all*) network è una rete neurale addestrata per supportare moltissime architetture diverse. Questo permette di ottenere moltissimi possibili sub-modelli tra i quali cercare quello desiderato che risponda al meglio alle nostre specifiche, tutto questo senza perdere accuratezza rispetto ad uno stesso modello addestrato appositamente.
L'OFA in considerazione (Tabella 3.12) ha performance migliori di algoritmi NAS comunemente usati quali *MobileNetV3* (fino a 4.0% in più nell'accuratezza top1) o *EfficientNet* (2.6 volte più veloce in termini di latenza).

Model	ImageNet Top1 (%)	MACs	Mobile latency	Search cost (GPU hours)	Training cost (GPU hours)	Total cost ($N = 40$)		
						GPU hours	CO_2e (lbs)	AWS cost
MobileNetV2 [31]	72.0	300M	66ms	0	$150N$	6k	1.7k	$18.4k
MobileNetV2 #1200	73.5	300M	66ms	0	$1200N$	48k	13.6k	$146.9k
NASNet-A [44]	74.0	564M	-	$48{,}000N$	-	1,920k	544.5k	$5875.2k
DARTS [25]	73.1	595M	-	$96N$	$250N$	14k	4.0k	$42.8k
MnasNet [33]	74.0	317M	70ms	$40{,}000N$	-	1,600k	453.8k	$4896.0k
FBNet-C [36]	74.9	375M	-	$216N$	$360N$	23k	6.5k	$70.4k
ProxylessNAS [4]	74.6	320M	71ms	$200N$	$300N$	20k	5.7k	$61.2k
SinglePathNAS [8]	74.7	328M	-	$288 + 24N$	$384N$	17k	4.8k	$52.0k
AutoSlim [38]	74.2	305M	63ms	180	$300N$	12k	3.4k	$36.7k
MobileNetV3-Large [15]	75.2	219M	58ms	-	$180N$	7.2k	1.8k	$22.2k
OFA w/o PS	72.4	235M	59ms	40	1200	1.2k	0.34k	$3.7k
OFA w/ PS	**76.0**	230M	58ms	40	1200	1.2k	0.34k	$3.7k
OFA w/ PS #25	**76.4**	230M	58ms	40	$1200 + 25N$	2.2k	0.62k	$6.7k
OFA w/ PS #75	**76.9**	230M	58ms	40	$1200 + 75N$	4.2k	1.2k	$13.0k
$\text{OFA}_{\text{Large}}$ w/ PS #75	**80.0**	595M	-	40	$1200 + 75N$	4.2k	1.2k	$13.0k

Figura 3.12 Confronto di prestazioni e cost tra modelli comunemente utilizzati e modelli OFA

3.9 Altre soluzioni

Oltre ai metodi presentati nei punti precedenti vi sono molti altri studi riguardo l'implementazione e l'ottimizzazione di algoritmi di machine learning su microcontrollori, alcuni esempi possibili sono:

- L'uso del transfer learning tra domini differenti (*Federeted Transfer Learning*), utilizzando, per esempio, un'implementazione chiamata *TinyFedTL*.

- L'uso di acceleratori hardware/software quali HBDCA (*High-Accuracy BRAM-Defined CNN Accelerator*) e LB-CNN (*Light Binary CNN*) per migliorare l'efficienza e la velocità della rete neurale.

- L'uso di framework alternativi per lo sviluppo e addestramento della rete sul microcontrollore, quali *Rosler* (*Random Sketch Learning*) e *TinyOL* (*TinyML with Online-Learning*).

- L'uso di algoritmi più intelligenti quali *RaScaNet*, che completa la lettura di un'immagine attraverso l'ausilio di una rete neurale ricorrente (RNN).

Capitolo 4

4. Prestazioni e applicazioni

4.1 Prestazioni

L'analisi delle prestazioni è un aspetto fondamentale della ricerca di nuove soluzioni ingegneristiche. Permette il confronto, la valutazione e il miglioramento delle prestazioni dei sistemi analizzati e costituisce uno strumento importante per garantire la longevità e il successo di un determinato campo di ricerca.
Per quanto riguarda il panorama del TinyML non vi sono ancora dei test riconosciuti in maniera diffusa.
Tra le varie possibilità quella più promettente nasce da un gruppo costituito da oltre 30 organizzazioni e 75 membri che prende il nome di *TinyMLPerf*.
Il test suddivide i dispositivi in due classi "closed" e "open", la prima richiede la standardizzazione dei modelli e dataset utilizzati, la seconda invece non pone vincoli sull'implementazione. I dispositivi vengono in seguito testati su tre parametri: accuratezza, latenza ed energia utilizzata.
Nonostante gli sforzi iniziali c'è ancora molta strada da fare, in particolare la grandissima varietà di microcontrollori disponibili rende difficile avere un'idea oggettiva delle prestazioni effettive senza avere a disposizione una grande quantità di dati.
In seguito vengono presentate alcune tabelle contenenti dei confronti tra dispositivi diversi.

						Benchmark Results							
						Task	Visual Wake Words		Image Classification		Keyword Spotting		Anomaly Deteciton	
						Data	Visual Wake Words Dataset		CIFAR-10		Google Speech Commands		ToyADMOS (ToyCar)	
						Model	MobileNetV1 (0.25x)		ResNet-V1		DSCNN		FC AutoEncoder	
						Accuracy	80% (top 1)		85% (top 1)		90% (top 1)		0.85 (AUC)	
Board Name	SoC Name	Processor(s) & Number	Accelerator(s) & Number	Software	Notes	Units	Latency in ms	Energy in uJ	Latency in ms	Energy in uJ	Latency in ms	Energy in uJ	Latency in ms	Energy in uJ
GAP9 EVK	GAP9	RISC-V Core (1+9)	NE16 (1)	GreenWaves GAPFlow	GAP9 (370MHZ, 0.8Vcore)		1.13	58.4	0.62	40.4	0.48	26.7	0.18	7.29
GAP9 EVK	GAP9	RISC-V Core (1+9)	NE16 (1)	GreenWaves GAPFlow	GAP9 (240MHZ, 0.65Vcore)		1.73	40.8	0.95	27.7	0.73	18.6	0.27	5.25
NRF5340DK	nRF5340	Arm® Cortex®-M33		microTVM using CMSIS-NN backend	128MHz		232.0		316.1		76.1		6.27	
NUCLEO-L4R5ZI	STM32L4R5ZIT6U	Arm® Cortex®-M4		microTVM using CMSIS-NN backend	120MHz, 1.8Vbat		301.2	15531.4	389.5	20236.3	99.8	5230.3	8.60	443.2
NUCLEO-L4R5ZI	STM32L4R5ZIT6U	Arm® Cortex®-M4		microTVM using native codegen	120MHz, 1.8Vbat		336.5	17131.6	389.2	21342.3	144.0	7950.5	11.7	633.7
B_U585I_IOT02A	STM32U585	Arm® Cortex®-M33		Plumerai Inference Engine 2022.09	160MHz		107.0		107.1		35.4		4.90	
CY8CPROTO-062-4343W	PSoC 62 MCU	Arm® Cortex®-M4		Plumerai Inference Engine 2022.09	150MHz		192.5		193.1		61.4		8.70	
DISCO-F746NG	STM32F746	Arm® Cortex®-M7		Plumerai Inference Engine 2022.09	216MHz		57.0		64.8		19.1		2.30	
NUCLEO-L4R5ZI	STM32L4R5ZIT6U	Arm® Cortex®-M4		Plumerai Inference Engine 2022.09	120MHz		208.6		173.2		71.7		5.60	
xG24-DK2601B	EFR32MG24	Arm® Cortex®-M33	Silicon Labs MVP(1) (78 MHz, 1.8V)	TensorFlowLite for Microcontrollers, CMSIS-NN, Silicon Labs Gecko SDK			111.6	1139.2	120.9	1234.7	36.3	401.9	5.43	47.3
NUCLEO-H7A3ZI-Q	STM32H7A3ZIT6Q	Arm® Cortex®-M7		X-CUBE-AI v7.3.0	280MHz, 3.3Vbat		50.7	7978.5	54.3	8707.3	16.8	2721.8	1.82	266.5
NUCLEO-L4R5ZI	STM32L4R5ZIT6U	Arm® Cortex®-M4		X-CUBE-AI v7.3.0	120MHz, 1.8Vbat		230.5	10066.6	226.9	10681.6	75.1	3371.7	7.57	323.0
NUCLEO-U575ZI-Q	STM32U575ZIT6Q	Arm® Cortex®-M33		X-CUBE-AI v7.3.0	160MHz, 1.8Vbat		133.4	3364.5	139.7	3642.0	44.2	1138.5	4.84	119.1
NDP9120-EVL	NDP120	M0 + HiFi	Syntiant Core 2 (98MHz, 1.1V)	Syntiant TDK	Syntiant Core 2 (98MHz, 1.1Vcore)		4.10	97.2	5.12	139.4	1.48	43.8		
NDP9120-EVL	NDP120	M0 + HiFi	Syntiant Core 2 (30MHz, 0.9V)	Syntiant TDK	Syntiant Core 2 (30MHz, 0.9Vcore)		12.7	71.7	16.0	101.8	4.37	31.5		

Figura 4.1 Confronto, nella piattaforma TinyMLPerf, tra dispositivi differenti della categoria closed

	Board Name	Processor, Flash (MB), SRAM, Clock (MHz)
MCUs	B1: Teensy 4.0	Cortex-M7, 2, 1MB, 600
	B2: STM32 Nucleo H7	Cortex-M7, 2, 1 MB, 480
	B3: Arduino Portenta	Cortex-M7+M4, 2, 1MB, 480
	B4: Feather M4 Express	Cortex-M4, 2, 192KB, 120
	B5: Generic ESP32	Xtensa LX6, 4, 520KB, 240
	B6: Arduino Nano 33	Cortex-M4, 1, 256KB, 64
	B7: Raspberry Pi Pico	Cortex-M0+, 16, 264KB, 133
	Name: Feature dimension, Class counts	
Datasets	D1: Iris Flowers: 4, 3	D6: Breast Cancer: 30, 2
	D2: Wine: 13, 3	D7: Texture: 40, 11
	D3: Vowel: 13, 11	D8: Drive Diagnosis, 48, 11
	D4: Silhouettes: 18, 4	D9: MNIST Digits: 64, 10
	D5: Anuran Calls: 22, 8	D10: Human Activity: 74, 6
	Name: Topology	
Fully Connected NNs	*FC 1x10*: 1 layer with 10 neurons	
	FC 10+50: 1^{st} layer with 10 neurons, $2^{nd}$with 50	
	FC 10x10: 10 layers, with 10 neurons in each layer	

Figura 4.2 Specifiche per la figura 4.3

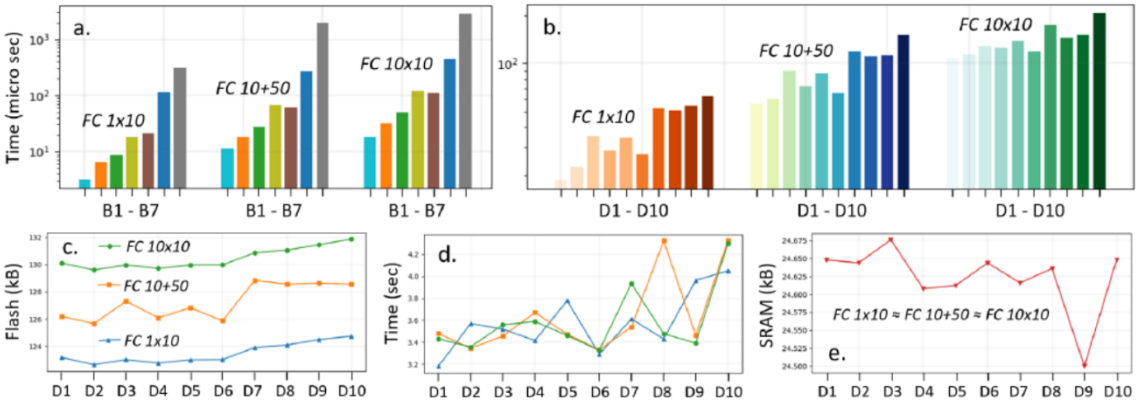

Figura 4.3 Risultati del benchmark su dispositivi e dataset differenti

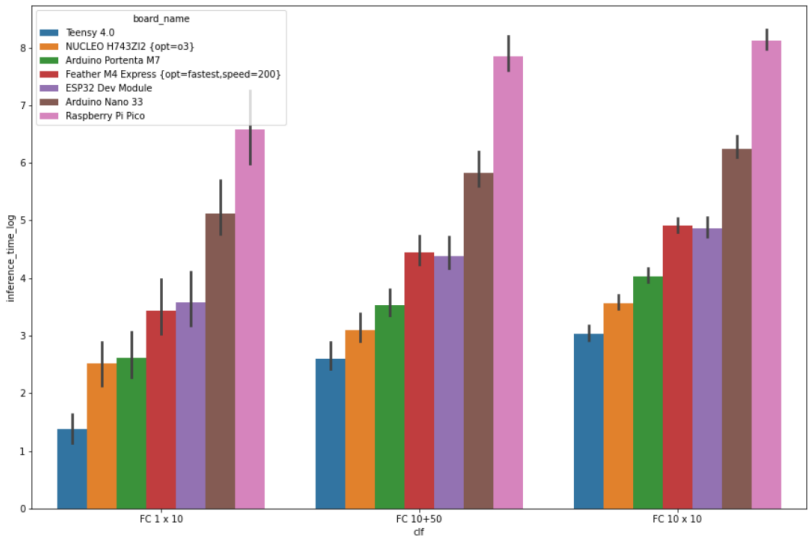

Figura 4.4 Confronto in scala logaritmica tra benchmark di vari microcontrollori utilizzando la stessa rete fully connected.
1x10 significa un solo livello da 10 neuroni, 10+50 sono due livelli rispettivamente da 10 e 50 neuroni, mentre 10x10 sono 10 livelli da 10 neuroni ciascuno.

4.2 Applicazioni

Sono moltissimi gli scenari applicativi che beneficiano dall'uso di un microcontrollore dotato di intelligenza artificiale e sono destinati ad aumentare col passare del tempo.
Ho scelto di presentarne alcuni esempi importanti:

- *Image recognition*: la maggior parte delle applicazioni di IA richiede il riconoscimento di determinati oggetti all'interno di un'immagine o video, da questo modello si espande un grandissimo numero di tecnologie, quali *face detection*, *gesture recognition* e *activity detection*.
- *Speech recognition*: ha diversi utilizzi, ad esempio per interfacciarsi con gli smartphones attraverso l'assistente (*Voice Activity Detection*).
- Guida autonoma: importante in particolare per tutti quei dispositivi dalle dimensioni limitate, ad esempio i droni o i veicoli radiocomandati.
- *Anomaly detection*: un paradigma generale che si riferisce al riconoscimento di un'anomalia rispetto alla distribuzione dei dati, potrebbe essere usato per esempio per rilevare incidenti, malfunzionamenti o problemi di salute.
- Applicazioni di studio e riconoscimento della flora e della fauna, dove il risparmio energetico e l'indipendenza da internet sono delle caratteristiche fondamentali.

Dalla combinazione e modifica di questi modelli possono nascere centinai di altre applicazioni possibili. È importante evidenziare come i microcontrollori, grazie ai loro bassi consumi e alle loro dimensioni contenute, aprano la strada ad una serie di oggetti che altrimenti non farebbero/faranno parte della nostra vita quotidiana.

Capitolo 5

Conclusioni

Risulta assolutamente evidente l'importanza che il TinyML riveste e ancor più rivestirà nel futuro delle nostre tecnologie. La possibilità di rendere intelligente una piccola parte di un oggetto permetterà la diffusione di strumenti che un tempo sarebbero stati troppo costosi o troppo ingombranti.
Considerando quanto recente sia lo sviluppo di questo campo dell'intelligenza artificiale è importante riconoscere quanto ancora ci sia da fare e migliorare, in particolare nella creazione di software e hardware più eterogenei e nello sviluppo di benchmark riconosciuti e standardizzati. Tutto ciò diventerà più semplice una volta che il TinyML avrà preso più spazio all'interno del vasto mondo dell'intelligenza artificiale. Al momento progetti che richiedono settimane di addestramento su supercomputer stanno attirando molta più attenzione, ma presto le persone inizieranno ad accorgersi delle potenzialità che il machine learning su microcontrollori possiede.

Bibliografia

[1] Gunther Gridling, Bettina Weiss. *Introduction to Microcontrollers*. Vienna University of Technology, 2007, pp. 1-2.

[2] Zhi-Hua Zhou, *Machine Learning*, Springer Nature Singapore, 2021, pp. 2, DOI: https://doi.org/10.1007/978-981-15-1967-3

[3] Thiyagarajan MK. "*The Exciting Future Potential of Machine Learning*". In Data Science Blogathon, 2021.

[4] Partha Pratim Ray. "*A review on TinyML: State-of-the-art and prospects*". Journal of King Saud University - Computer and Information Sciences, 2022. DOI: https://doi.org/10.1016/j.jksuci.2021.11.019

[5] R. Banbury et al. "*Benchmarking Tinyml Systems: Challenges and Direction*", arXiv, 2021, https://arxiv.org/pdf/2003.04821.pdf

[6] "*Automated Machine Learning (ML) tool for STM32 developers*" in https://www.st.com/en/development-tools/nanoedgeaistudio.html#overview

[7] "Implement AI on Any Hardware With AIfES" in https://www.ims.fraunhofer.de/en/Business-Unit/Industry/Industrial-AI/Artificial-Intelligence-for-Embedded-Systems-AIfES.html

[8] E. Hassan, Y. Halawani, B. Mohammad and H. Saleh, "*Hyper-Dimensional Computing Challenges and Opportunities for AI Applications*," in IEEE Access, vol. 10, pp. 97651-97664, 2022, DOI: 10.1109/ACCESS.2021.3059762.

[9] Hongyu Miao, Felix Xiaozhu Lin. "*Enabling Large Neural Networks on Tiny Microcontrollers with Swapping*", arXiv, 2021, https://arxiv.org/pdf/2101.08744v3.pdf

[10] Wong et al. "*TinySpeech: Attention Condensers for Deep Speech Recognition Neural Networks on Edge Devices*", arXiv, 2020, https://arxiv.org/pdf/2008.04245v6.pdf

[11] Edgar Liberis, Łukasz Dudziak, Nicholas D. Lane, "*µnas: Constrained Neural Architecture Search for Microcontrollers*", arXiv, 2021, https://arxiv.org/pdf/2010.14246v3.pdf

[12] Signoretti, G.; Silva, M.; Andrade, P.; Silva, I.; Sisinni, E.; Ferrari, P. "*An Evolving TinyML Compression Algorithm for IoT Environments Based on Data Eccentricity*". Sensors 2021, 21, 4153. https://doi.org/10.3390/s21124153

[13] Thakker et al. "*Compressing Language Models using Doped Kronecker Products*", arXiv, 2020, https://arxiv.org/pdf/2001.08896v5.pdf

[14] Hu et al. "*Starfish: Resilient Image Compression for AIoT Cameras*", 2020, DOI: https://panhu.me/pdf/2020/Starfish.pdf

[15] Zhuo et al. "*An Empirical Study of Low Precision Quantization for TinyML*", arXiv, 2022, https://arxiv.org/pdf/2203.05492v1.pdf

[16] Ghamari et al. "*Quantization-Guided Training for Compact TinyML Models*", arXiv, 2021, https://arxiv.org/pdf/2103.06231v1.pdf

[17] Cai et al. "*Once-For-All: Train One Network and Specialize it for Efficient Deployment*", arXiv, 2020, https://arxiv.org/pdf/1908.09791v5.pdf

[18] Mathur et al. "*On-device federated learning with Flower*", arXiv, 2021, https://arxiv.org/pdf/2104.03042v1.pdf

[19] Sudharsan et al. (2021). "*TinyML Benchmark: Executing Fully Connected Neural Networks on Commodity Microcontrollers*". 10.1109/WF-IoT51360.2021.9595024.

[20] https://mlcommons.org/en/inference-tiny-10/

[21] Li et al. "*SWIS - Shared Weight bIt Sparsity for Efficient Neural Network Acceleration*", arXiv, 2021, https://arxiv.org/pdf/2103.01308v2.pdf

[22] Simone, "TinyML Benchmark: Fully Connected Neural Networks (now with Raspberry Pi Pico!)", 2021, DOI: https://eloquentarduino.github.io/2021/04/tinyml-benchmark-fully-connected-neural-networks/