Valutare i limiti pratici di TinyML: un approccio sperimentale

Giovanni Delnevo
Dipartimento di Informatica e Ingegneria Università di Bologna Bologna, Italia
giovanni.delnevo2@unibo.it

Catia Prandi
Dipartimento di Informatica e Ingegneria Università di Bologna Bologna, Italia
catia.prandi2@unibo.it

Silvia Mirri
Dipartimento di Informatica e Ingegneria Università di Bologna Bologna, Italia silvia.mirri@unibo.it

Pietro Manzoni
Dipartimento di Ingegneria Informatica Politecnico di Valencia
Valencia, Spagna
pmanzoni@disca.upv.es

Riassunto—Tiny Machine Learning (TinyML) è un nuovo campo di ricerca che apre la possibilità di incorporare l'intelligenza locale in oggetti frugali creando così nuove opportunità per costruire "reti di intelligenza collettiva". La riduzione dei consumi energetici è probabilmente il motivo principale per cui TinyML merita attenzione in termini di sostenibilità, ma non è l'unico. Anche i bassi costi dell'hardware e la possibilità di aumentare il livello di sicurezza dei dati e la privacy degli utenti sono motivi eccezionali. In questo lavoro, presentiamo i risultati di un'analisi di sensitività che abbiamo condotto per valutare le prestazioni di una Random Forest con i dati raccolti da un dispositivo hardware all'avanguardia. Ci siamo concentrati sulla valutazione della sensibilità del rilevamento di suoni, colori e modelli di vibrazioni. I risultati mostrano che TinyML può essere assolutamente utilizzato per discriminare correttamente tra diverse gamme di suoni, colori e modelli di vibrazioni, aprendo la strada allo sviluppo di nuove promettenti applicazioni sostenibili.

Termini dell'indice: TinyML, sostenibilità, Random Forest, analisi di sensibilità

I. INTRODUZIONE

Tiny Machine Learning (TinyML) è un nuovo campo di ricerca che mira a portare le tecniche di apprendimento automatico alla portata di dispositivi molto piccoli e applicazioni di sistemi embedded. In particolare, nell'ambito del lavoro che presentiamo, stiamo considerando dispositivi a basso costo (MicroController Unit - MCU) che possono lavorare a 1 mW di potenza [1] integrando una CPU di base, una ridotta quantità di memoria e alcune alternative di connettività di base. In combinazione con l'Edge Computing [2], TinyML consente al calcolo di essere distribuito lungo il percorso dal dispositivo finale al core di Internet, riducendo così la necessità di data center più affamati di energia che sono tipicamente utilizzati per l'elaborazione di applicazioni di Machine Learning nelle piattaforme cloud [3], [4].

La riduzione dei consumi energetici è probabilmente il motivo principale per cui TinyML merita attenzione in termini di sostenibilità, anche se non è l'unico. In effetti, anche i bassi costi dell'hardware sono un aspetto critico quando si implementa un gran numero di dispositivi. Inoltre, l'esecuzione dell'inferenza ML sul dispositivo aumenta il livello di sicurezza dei dati e privacy degli utenti [5]. I dati possono essere sensibili e l'elaborazione e l'analisi sul dispositivo non richiede l'invio su Internet. Infine, il calcolo all'edge riduce i requisiti di larghezza di banda della rete, consentendo l'uso di tecnologie di comunicazione ad alta efficienza energetica [6].

TinyML presenta opportunità uniche anche per oggetti frugali [7], che sono penetrati nella nostra vita quotidiana grazie alla rapida comparsa dell'IoT (Internet of Things). La possibilità di incorporare l'intelligenza locale in oggetti frugali apre nuove opportunità per costruire “reti di intelligenza collettiva”. L'innovazione frugale consiste in un approccio creativo e orientato ai problemi alla risoluzione dei problemi che parte dalle esigenze degli utenti e lavora dal basso verso l'alto per sviluppare soluzioni contestualmente appropriate [8]. Tale processo è strettamente connesso alla sostenibilità, come evidenziato da Khan in [9]. Secondo i risultati riportati in quel documento, l'innovazione frugale può essere un fattore trainante, non solo nel raggiungimento della sostenibilità sociale, ma anche nella promozione degli Obiettivi di Sviluppo Sostenibile. Per questo motivo, l'integrazione di TinyML all'interno di oggetti frugali può arricchire significativamente l'attuale panorama delle applicazioni degli oggetti frugali.

Sanchez-Iborra e Skarmeta hanno riassunto cinque possibili contesti applicativi che potrebbero trarre vantaggio dall'integrazione di TinyML in oggetti frugali [7]. Il primo è l'eHealth, in cui i dispositivi finali intelligenti potrebbero essere utilizzati per il monitoraggio della salute, l'assistenza visiva, gli apparecchi acustici e il rilevamento personale, solo per citarne alcuni [10]. Il secondo è costituito da spazi intelligenti (ad es. città intelligenti ed edifici cognitivi) in cui tali sistemi potrebbero essere facilmente implementati per prendere decisioni decentralizzate e rapide [11]. Il terzo campo di applicazione sono i servizi veicolari. In particolare, i sistemi di trasporto intelligenti e cooperativi potrebbero essere impiegati anche su dispositivi per la mobilità personale come biciclette condivise, ciclomotori elettrici e scooter [12]. Un altro contesto applicativo è Industry 4.0 in cui TinyML potrebbe consentire di aggiungere una qualche forma di intelligenza lungo la catena di produzione per perfe

processi produttivi e per tracciare gli asset [13]. Infine, gli ultimi settori sono l'agricoltura intelligente e l'agricoltura, dove TinyML potrebbe migliorare i processi di monitoraggio e controllo intelligenti con lo scopo di aumentare l'efficienza e la salute delle colture e degli animali [14].

In questo documento, presentiamo un'analisi di sensibilità dei limiti pratici di TinyML utilizzando un approccio sperimentale.
Per mezzo di un dispositivo di uso comune, un “Arduino Nano 33 BLE sense”, che supporta una serie di sensori incorporati, abbiamo valutato il livello di precisione ottenibile nel rilevamento di suoni, colori e vibrazioni. L'obiettivo è definire meglio l'effettiva classe di applicazione che può essere progettata con l'attuale stato dell'arte dell'hardware combinato con TinyML. La nostra metodologia si basava sulla raccolta del set di dati direttamente utilizzando i sensori Arduino Nano. Quindi, utilizzando questi set di dati, abbiamo addestrato un algoritmo di apprendimento automatico per classificare determinate proprietà della variabile misurata. Infine, abbiamo testato le sue prestazioni di rilevamento direttamente sul dispositivo. I risultati mostrano che, utilizzando un metodo Random Forest, è possibile discriminare tra diverse gamme di frequenze, colori e modelli e intensità di vibrazioni.

Al meglio delle nostre conoscenze, nessun lavoro in letteratura ha affrontato l'analisi di sensibilità della combinazione dei sensori del microcontrollore con algoritmi di machine learning che effettuano l'inferenza direttamente su questi dispositivi. Insieme al rilascio di diversi stack software come Tensorflow lite micro [15], Edge2train [16], OpenNN [17], libreria Edge Machine Learning [18] e Resource Constrained Edge-Neural Networks [19], sono stati proposti diversi benchmark .

Tra questi, possiamo citare MLPerf Tiny Benchmark [20], CoreMark [21] e TinyML Benchmark [22]. Ad ogni modo, lo scopo di tali benchmark è misurare alcuni aspetti nel processo di inferenza degli algoritmi di apprendimento automatico come accuratezza, latenza e consumo di energia.

Il promemoria del documento è organizzato come segue. La Sezione II descrive in dettaglio ogni fase del nostro approccio, descrivendo come abbiamo condotto la raccolta dei dati, l'addestramento del modello e la sua valutazione. Quindi, nella Sezione III, descriviamo i risultati dell'analisi sulla sensibilità dei sensori nel rilevare la frequenza, il colore e le vibrazioni. Infine, la Sezione IV chiude il lavoro, evidenziando alcuni lavori futuri.

II. METODOLOGIA

In questa sezione, descriviamo in dettaglio il nostro approccio, composto da tre fasi principali: raccolta dei dati, training del modello e valutazione del modello. Lo abbiamo seguito per ogni aspetto analizzato: suoni, colori e pattern di vibrazioni. Ciascuna fase sarà dettagliata separatamente nelle sottosezioni seguenti.

A. Dati e raccolta

Il processo di raccolta dei dati è stato effettuato utilizzando i diversi sensori montati su Arduino Nano 33 BLE.

Per generare le frequenze, abbiamo utilizzato l'app mobile Android Frequency Generator1 . Permette di generare frequenze da 1Hz (infrasuoni) a 22.000Hz (ultrasuoni), utilizzando frequenze singole o multiple con diverse onde sonore (sinusoidali, quadrate, triangolari e a dente di sega). Per registrare tali frequenze abbiamo utilizzato il microfono digitale MP34DT05. Registra 256 letture in modo continuo con una frequenza di campionamento di 16 KHz. Viene applicata una modulazione della densità di impulso per ottenere 128 valori, di cui viene calcolato il RMS (Root Mean Square). Viene applicata una soglia per registrare solo quando è presente il suono ed evitare così il silenzio o il rumore di fondo. Quindi, se l'RMS è superiore a una soglia fissa, inizia il processo di registrazione. Consiste semplicemente nell'ottenere 32 valori RMS, con un ritardo di 20 ms tra ciascun campione, per un periodo totale di 640 ms. Questa matrice di 32 valori viene utilizzata come rappresentazione dei suoni registrati.
Per ogni frequenza considerata, abbiamo raccolto 100 campioni.

Per quanto riguarda i colori, abbiamo considerato alcune sfumature di colore leggermente diverse sia stampate su carta che visualizzate su schermo. Abbiamo utilizzato il sensore APDS-9960, un sensore digitale di prossimità e luce ambientale in grado di recuperare anche il colore. Restituisce le tre componenti RGB, più un valore relativo all'intensità della luce ambientale. A questi quattro abbiamo aggiunto anche la prossimità, poiché potrebbe influenzare notevolmente il colore. Quindi, ogni campione è composto da cinque numeri interi. Per ogni colore abbiamo raccolto 200 campioni.

Infine, per generare le vibrazioni, abbiamo sfruttato l'app mobile Android Vibrate Pattern Maker2 . Permette di generare pattern alternando periodi di vibrazione e “silenzio”. Per ogni periodo è possibile specificarne la durata in millisecondi ma non è possibile variare l'intensità delle vibrazioni. Tali schemi possono essere ripetuti all'infinito.
Abbiamo generato diversi modelli variando vibrazioni e periodi di silenzio e abbiamo cercato di discriminarli. Quindi, abbiamo anche cercato di discriminare tra modelli di vibrazioni di diversa intensità. Abbiamo utilizzato un motore di vibrazione con cinque diverse impostazioni di velocità, che abbiamo utilizzato per generare semplici schemi di vibrazione variando solo l'intensità delle vibrazioni. Per rilevare le vibrazioni, abbiamo utilizzato il sensore LSM9DS1 che fornisce un sensore di accelerazione. Il sensore restituisce le accelerazioni nelle tre dimensioni x, yez. Abbiamo raccolto 20 campioni, con un ritardo di 50 ms tra ogni campione. Questa matrice di 60 valori (20 campioni di tre valori) viene utilizzata come rappresentazione del modello di vibrazione registrato e copre 1 secondo di dati.

È importante notare che i dati vengono raccolti in condizioni di laboratorio. Le applicazioni reali porranno problemi e sfide. Ma il nostro scopo, qui, è solo quello di condurre un'analisi di sensibilità dei limiti pratici di TinyML utilizzando un approccio sperimentale

B. Formazione del modello

Una volta raccolti dati sufficienti, si passa al training del modello di machine learning. La fase di training è stata condotta su un PC generico e non su Arduino Nano. Abbiamo usato il linguaggio Python e abbiamo sfruttato la libreria Scikit-learn [23].

Abbiamo usato le foreste casuali in tutti gli esperimenti. Una foresta casuale è un meta stimatore che combina la previsione di

1play.google.com/store/apps/details?id=com.boedec.hoel.frequencygenerator

2play.google.com/store/apps/details?id=com.domago.vibratepatternmaker

diversi alberi decisionali su diversi sottocampioni della formazione
impostato con l'obiettivo di migliorare l'accuratezza della previsione e
prevenire il sovraffollamento. I modelli, che abbiamo impiegato nel
esperimenti, combinati 50 alberi decisionali. Il guadagno di informazioni
è stato valutato utilizzando l'impurità di Gini e non abbiamo impostato a
profondità massima dell'albero. Tutti gli altri parametri, come
il numero minimo di campioni richiesto per dividere un interno
nodo e il numero minimo di campioni richiesto per essere a
un nodo foglia, vengono lasciati con i loro valori predefiniti.

Abbiamo deciso di utilizzare le foreste casuali per diversi motivi [24].
Innanzitutto, li abbiamo preferiti alle reti neurali poiché le funzionalità
sono già disponibili e non è necessario estrarre/scoprire
loro e data la numerosità dei dati. Secondo, con
rispetto ad altri metodi di ensemble, come l'aumento del gradiente
e xgboost, abbiamo scelto solo un metodo che si è dimostrato
efficace in scenari reali [25], lasciando il confronto con
altri metodi come lavoro futuro.

In ogni esperimento, il set di dati è diviso in due parti.
L'80% dei campioni viene utilizzato per la formazione mentre il
il restante 20% viene utilizzato per la convalida. I dati sono divisi in a
modo stratificato con l'obiettivo di mantenere la stessa proporzione
per ogni classe. È importante ricordare che non abbiamo impiegato
un set di test poiché abbiamo valutato direttamente l'accuratezza del modello
su Arduino Nano. Inoltre, non ne abbiamo condotto nessuno
fase di tuning degli iperparametri del modello. Il principale
motivo è che di solito questa fase permette ai più fortunati
casi per migliorare la precisione di alcuni punti percentuali.
Dal momento che siamo interessati a comprendere la sensibilità generale
di modelli di machine learning implementati su Arduino Nano,
abbiamo deciso di non farlo.

C. Valutazione del modello

Una volta addestrati i modelli, ne abbiamo valutato le prestazioni. La
scelta di valutare il modello direttamente sul
Arduino Nano deriva dal fatto che non vogliamo
per trovare la massima precisione possibile utilizzando i dati
rileva attraverso i suoi sensori. Invece, in una prospettiva TinyML,
vogliamo valutare la massima accuratezza ottenibile da
un modello di apprendimento automatico in esecuzione su di esso. Questo è perché
avendo Arduino Nano risorse computazionali limitate,
è possibile che le prestazioni su di esso differiscano da quella
ottenuto su un dispositivo più potente come un PC. Importare
i modelli addestrati su Arduino Nano, abbiamo sfruttato il
Libreria EloquentTinyML3 . È una libreria che semplifica il
distribuzione di alcuni modelli Tensorflow Lite e Scikit-learn,
come reti neurali, foreste casuali e gradiente estremo
potenziamento. Una volta implementati, abbiamo utilizzato i modelli per prevedere il
classe di più input e, al termine del processo, we
ne ha valutato le prestazioni utilizzando diversi parametri. In particolare,
abbiamo sfruttato la precisione, il richiamo, il punteggio f1 e l'accuratezza.
Non abbiamo riportato l'accuratezza sul set di test calcolato su a
PC generico, poiché i risultati sono stati molto simili, con
una differenza massima dell'1%.

3github.com/eloquentarduino/EloquentTinyML

III. RISULTATI

In questa sezione presentiamo i risultati degli esperimenti
condotto per valutare la sensibilità di una foresta casuale in
rilevando la frequenza, il colore e le vibrazioni, misurate dal
Sensori Arduino Nano.

A. Sensibilità di rilevamento della frequenza

Abbiamo iniziato gli esperimenti per capire la sensibilità
del microfono valutando frequenze da 500Hz a
3.000 Hz, variandoli di 500 Hz. Il modello distribuito era
testato raccogliendo 20 esempi per ciascuna frequenza. Il
i risultati sono riportati nella Tabella I. Come mostrato, il modello è in grado di farlo
differenziano quasi perfettamente le frequenze analizzate.

TABELLA I
PRESTAZIONI DEL CLASSIFICATORE NEL RILEVAMENTO DELLE FREQUENZE DI VARIANDOLI DI 500HZ, DA 500HZ A 3.000HZ

Precisione di classe	Recall	Precisione del punteggio F1	
500	1.00 1.00 1.00 1.00	1.00	
1000	1.00 1.00 1.00 1.00	1.00	
1500	1.00 0.95 1.00	1.00	0,99
2000		1.00	
2500		0,97	
3000		0,98	

Quindi, abbiamo deciso di concentrarci sui 2.000 Hz e i 3.000 Hz
gamma di frequenza e per ridurre il passo da 500Hz a 200Hz.
Come nell'esperimento precedente, abbiamo valutato il dispiegato
modello su un set di test contenente 20 campioni per ogni classe.
La tabella II illustra le prestazioni del classificatore. È nel complesso
la precisione, che passa da 0,99 a 0,78, è comunque buona, anche se lo è
ha qualche problema nel distinguere 2.200Hz da 2.400Hz
e 2.800Hz da 3.000 mentre riconosce perfettamente 2.000Hz
e 2.600 Hz. Questo è visibile anche nella matrice di confusione, quella
abbiamo rappresentato in Figura 1.

TABELLA II
PRESTAZIONI DEL CLASSIFICATORE NEL RILEVAMENTO DELLE FREQUENZE DI VARIANDOLI DI 200HZ, DA 2.000HZ A 3.000HZ

Precisione di classe	Recall	Precisione del punteggio F1	
2000 1,00 1,00 2200 0,74 0,70		1.00	
2400 0,71 0,75 2600 0,90 1,00		0,72	
0,78 0,35		0,73	0,78
		1.00	
		0,71	
		0,48	

Infine, abbiamo ulteriormente ridotto la variazione di frequenza da
Da 200 Hz a 100 Hz. Parallelamente alle prove precedenti, in questo
anche uno abbiamo usato 20 esempi per ogni classe per valutare il
classificatore. In questo caso, come evidenziato dai risultati in Tabella
III, il modello non è più in grado di distinguere tra i
frequenze diverse. È interessante notare che lo è
ottime prestazioni sulle classi 2.000Hz e 2.100Hz.
In tutti gli altri casi, peggiorano notevolmente come in
il caso delle frequenze 2.200Hz e 2.400Hz, che ha il
i peggiori punteggi di F1 tra tutti, che sono rispettivamente 0,36 e 0,37.
Come visibile in Figura 2, oltre a non distinguere bene
gli esempi tra queste due classi, come era il caso in

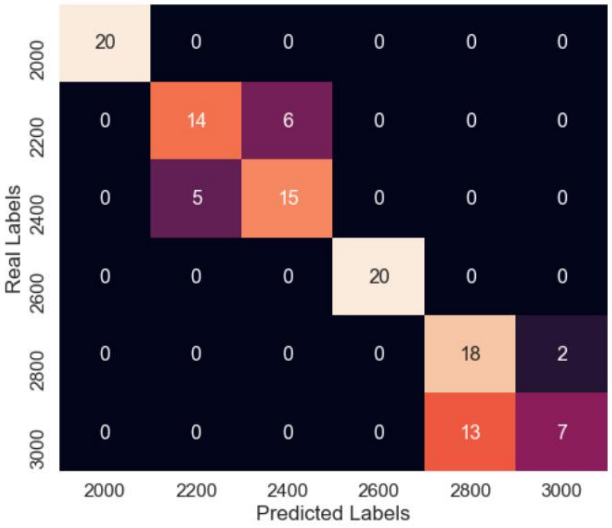

Fig. 1. Matrice di confusione ottenuta variando la frequenza di 200Hz, da Da 2.000 Hz a 3.000 Hz

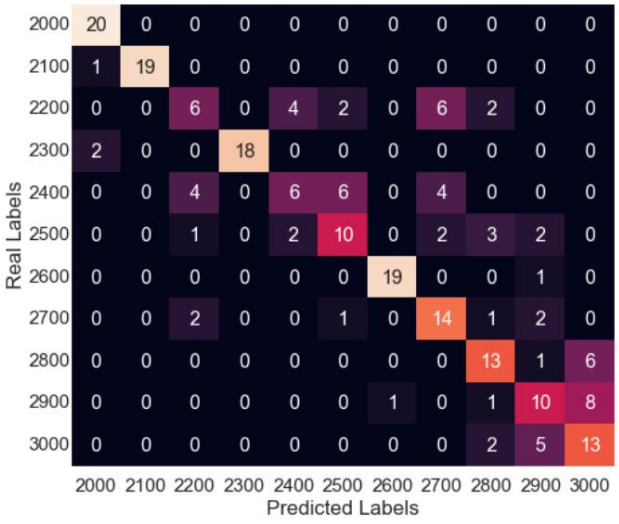

Fig. 2. Matrice di confusione ottenuta variando la frequenza di 100Hz, da Da 2.000 Hz a 3.000 Hz

l'esperimento precedente, ora anche i loro esempi sono confusi con quelli delle classi 2.500Hz e 2.700Hz.

TABELLA III
PRESTAZIONI DEL CLASSIFICATORE NEL RILEVAMENTO DELLE FREQUENZE DI VARIANDOLI DI 100HZ, DA 2.000HZ A 3.000HZ

Precisione di classe Recall Precisione del punteggio F1				
2000 0.87 1.00 2100 1.00 0.95			0,93	
2200 0.46 0.30 2300 1.00 0.38 2500			0,97	
0.53 0.50 2600 0.95 0.95 2700 0.86			0,36	
0.50 2900 0.48 0.65 3000 0.48			0,95	
			0,37	
			0,51	0,67
			0,95	
			0,61	
			0,62	
			0,49	
			0,55	

B. Sensibilità di rilevamento del colore

Per valutare la sensibilità del sensore di colore, abbiamo deciso di farlo impiegano sette diverse, ma molto simili, sfumature di verde. I loro I valori RGB esadecimali sono i seguenti: #229658, #30A161, #3DAB6B, #49B675, #55C17F, #60CC89 e #6BD793. Abbiamo rappresentato dei quadrati pieni di questi colori nella figura 3.

Abbiamo iniziato i nostri esperimenti visualizzando i quadrati su a tenere sotto controllo. In questi casi, i valori restituiti dal sensore sono molto preciso e non c'è bisogno di usare l'apprendimento automatico algoritmi per distinguerli. In realtà, un semplice set di affermazioni condizionali è sufficiente per raggiungere il 100% di esattezza. La situazione si complica considerando i colori stampati su carta. In questo scenario, la qualità del stampante e l'assenza di retroilluminazione, presente nella monitorare, hanno sicuramente un'influenza significativa. I risultati di i modelli che utilizzano tutte le sfumature di verde sono riportati in Tabella IV. Anche se le diverse tonalità sono molto simili, il sistema

Fig. 3. Scala di colore verde utilizzata negli esperimenti.

è in grado di distinguerli con una precisione accettabile. Il Il punteggio F1 nelle diverse classi varia in modo significativo, passando dallo 0.93 del #6BD793 fino a raggiungere lo 0.63 del #30A161.

Abbiamo anche condotto un ulteriore test per capire se il le prestazioni migliorano utilizzando colori meno simili. Noi semplicemente mantenuto solo tre colori. Gli estremi della scala verde, #229658 e #6BD793, e quello tra #49B675. In tal caso, la precisione del modello sale al 99%. Il model commette un solo errore, classificando erroneamente un campione di #49B675 come #6BD793.

TABELLA IV
PRESTAZIONI DEL CLASSIFICATORE NELLA RILEVAZIONE DEI COLORI

Classe	Precisione	Richiamo della precisione	del punteggio F1	
#229658	0,78	0,70	0,74	
#30A161	0,63	0,62	0,63	
#3DAB6B	0,72	0,84	0,78	
#49B675	0,88	0,88	0,88	0,79
#55C17F	0,80	0,72	0,76	
#60CC89	0,75	0,84	0,79	
#6BD793	0,96	0,90	0,93	

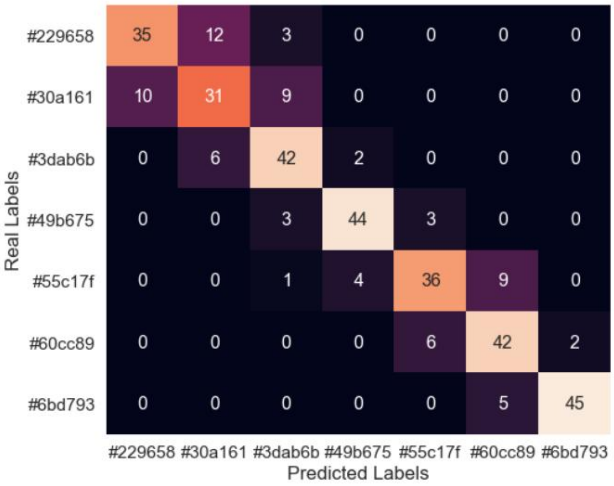

Fig. 4. Matrice di confusione ottenuta durante il rilevamento dei colori

TAVOLA V
PRESTAZIONI DEL CLASSIFICATORE NEL RILEVAMENTO DEI MODELLI DI VIBRAZIONE DI 100MS, 150MS E 200MS

Classe	Precisione	Richiamo	della precisione	del punteggio F1
100 ms - 100 ms	1	1	1	1
150 ms - 150 ms	1	1		
200 ms - 200 ms	1	1	1 1	

C. Sensibilità di rilevamento delle vibrazioni

Infine, abbiamo condotto alcuni esperimenti per valutare il sensibilità del modello distribuito su Arduino Nano in rilevamento delle vibrazioni. Abbiamo iniziato a cercare di riconoscere tre diversi modelli in cui le vibrazioni durano come la pausa periodo, rispettivamente 100 ms, 150 ms e 200 ms. Il modello è in grado di discriminare perfettamente tra i diversi modelli, come testimoniano le metriche riportate in Tabella V.

Quindi, abbiamo aggiunto tre ulteriori pattern che durano rispettivamente 10 ms, 25 ms e 50 ms. I risultati non cambiano, con il modello che è ancora in grado di discriminare perfettamente tra i modelli diversi. I risultati sono riportati nella Tabella VI.

TABELLA VI
PRESTAZIONI DEL CLASSIFICATORE NEL RILEVAMENTO DEI MODELLI DI VIBRAZIONE DI 10MS, 25MS, 50MS, 100MS, 150MS E 200MS

Classe	Precisione	Richiamo	della precisione	del punteggio F1
10ms -10ms	1	1	1	1
25ms - 25ms	1	1	1	
50ms - 50ms	1	1	1	
100ms - 100ms	1	1	1	
150ms - 150ms	1	1	1	
200ms - 200ms	1	1	1	

TABELLA VII
PRESTAZIONI DEL CLASSIFICATORE NEL RILEVAMENTO DEI MODELLI DI VIBRAZIONE DI 25MS E 50MS CON PAUSE DI 50MS E 25MS

Classe	Precisione	Richiamo	della precisione	del punteggio F1
25ms - 50ms			1	1
50ms - 25ms	1 1	1 1	1	

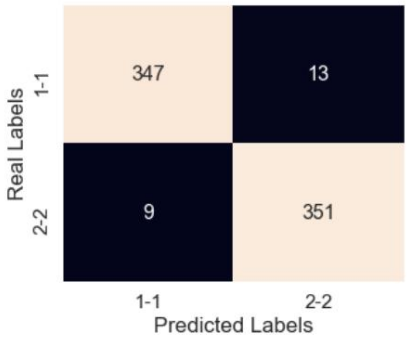

Fig. 5. Matrice di confusione ottenuta rilevando un pattern di vibrazioni di 1 ms e 2ms

TABELLA VIII
PRESTAZIONI DEL CLASSIFICATORE NEL RILEVAMENTO DEI MODELLI DI VIBRAZIONE DI 1MS E 2MS

Classe	Precisione	Richiamo	della precisione	del punteggio F1
1ms - 1ms	0,97	0,96	0,97	0,97
2ms - 2ms	0,96	0,97	0,97	

Abbiamo anche valutato due pattern con durata diversa di vibrazioni e pause. Il primo composto da vibrazioni di 25 ms seguiti da pause di 50 ms. Nella seconda, invece, le vibrazioni durano 50 ms mentre le pause durano 25 ms. Ancora una volta, il modello rileva perfettamente i due modelli, come mostrato nella tabella VII.

Nell'ultimo test effettuato, abbiamo portato gli schemi agli estremi con due modelli di vibrazione uguale e periodi di pausa di rispettivamente 1ms e 2ms. Comunque, le prestazioni restano eccellente, come riportato in Tabella VIII, con solo 22 esempi classificato erroneamente, come mostrato nella Figura 5.

Per quanto riguarda il rilevamento di diverse intensità di vibrazione, noi ha semplicemente cercato di riconoscere le impostazioni a cinque velocità dei motori a vibrazione. Anche in questo caso abbiamo ottenuto ottime prestazioni, con il classificatore che raggiunge il 99% di precisione, come mostrato in Tabella IX e Figura 6.

TABELLA IX
PRESTAZIONI DEL CLASSIFICATORE NELLA RILEVAZIONE DELLE VIBRAZIONI

Precisione di classe	Recall	Precisione del punteggio F1		
	1.00	0,99	1.00	0,99
1	0.99	1,00	1.00	
2	1.00	0,99	1.00	
3	0.99	0,97	0,98	
4 5	0.97	0,99	0,98	

IV. CONCLUSIONE E LAVORI FUTURI

TinyML sta guadagnando una crescente attenzione sia nel mondo accademico che industria, a causa dei vantaggi legati alla macchina da corsa inferenza di apprendimento su dispositivi a bassa potenza. Le sue possibili applicazioni in diversi contesti sono numerose e, in effetti, il rilevamento ambientale sostenibile è uno dei più promettenti. In questo lavoro, abbiamo valutato la sensibilità di una foresta casuale addestrato con i dati raccolti da Arduino Nano 33 BLE sensori. Abbiamo sperimentato la rilevazione di suoni, colori, e pattern di vibrazioni. I risultati mostrano che TinyML può esserlo

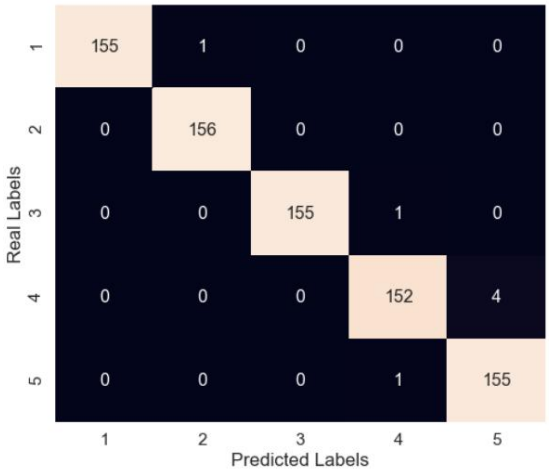

Fig. 6. Matrice di confusione ottenuta rilevando le vibrazioni generate dalle 5 impostazioni di velocità del frullatore ad immersione

assolutamente utilizzato per discriminare correttamente tra diverse gamme di suoni, colori e pattern di vibrazioni, aprendo la strada allo sviluppo di nuove promettenti applicazioni sostenibili.
Anche se è importante ricordare che i dati utilizzati negli esperimenti sono stati raccolti in condizioni di laboratorio.

I lavori futuri sono molteplici. In primo luogo, è possibile condurre un'analisi multidimensionale dei compromessi di altri aspetti, come il consumo di energia, il tasso di inferenza e i tassi di occupazione della memoria, insieme a un'analisi delle differenze matematiche tra gli algoritmi di apprendimento di implementazioni sia minuscole che non minuscole. Si potrebbe quindi effettuare un confronto con altre strategie di enembling, come in [26].

RINGRAZIAMENTI

Questo lavoro è stato parzialmente sostenuto dalla “Conselleria de Educacion, Investigacion, Cultura y Deporte, Direccion General de Universidades e Investigacion” della Comunitat Valenciana, Spagna, nell'ambito del Grant AICO/2020/302.

RIFERIMENTI

[1] P. Warden e D. Situnayake, Tinyml: Machine learning con tensorflow lite su arduino e microcontrollori a bassissima potenza. O'Reilly Media, 2019.

[2] K. Nakamura, P. Manzoni, M. Zennaro, J.-C. Cano, CT Calafate e JM Cecilia, "Fudge: A frugal edge node for advanced iot solutions in contexts with limited resources", in Atti del 1° Workshop sulle esperienze con la progettazione e l'implementazione di frugal Smart Objects, (New York, NY , Stati Uniti), pag. 30–35, Associazione per le macchine informatiche, 2020.

[3] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina e P. Whatmough, “Micronet: architetture di rete neurali per la distribuzione di tinyml applicazioni sui microcontrollori delle materie prime,” Proceedings of Machine Learning and Systems, vol. 3, 2021.

[4] H. Doyu, R. Morabito e M. Brachmann, "Un ecosistema tinymlaas per l'apprendimento automatico in iot: panoramica e sfide di ricerca", nel 2021 Simposio internazionale sulla progettazione, automazione e test VLSI (VLSI DAT), pp. 1–5, IEEE, 2021.

[5] MS Islam, H. Verma, L. Khan e M. Kantarcioglu, "Sistema di gestione dei dati iot eterogeneo in tempo reale sicuro", nel 2019 Prima IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS- ISA), pagg. 228–235, IEEE, 2019.

[6] W. Mao, Z. Zhao, Z. Chang, G. Min e W. Gao, "Energy efficient industrial internet of things: Overview and open issues", IEEE Transactions on Industrial Informatics, 2021.

[7] R. Sanchez-Iborra e AF Skarmeta, "Oggetti intelligenti frugali abilitati Tinyml: sfide e opportunità", IEEE Circuits and Systems Magazine, vol. 20, n. 3, pagg. 4–18, 2020.

[8] H. Kroll, M. Gabriel, A. Braun, F. Engasser, M. Meister ed E. Muller, "Studio sull'innovazione frugale e la reingegnerizzazione delle tecniche tradizionali", 2015.

[9] R. Khan, "Come l'innovazione frugale promuove la sostenibilità sociale", Sus sostenibilità, vol. 8, n. 10, pag. 1034, 2016.

[10] P. Randhawa, V. Shanthagiri e A. Kumar, "Una revisione sull'apprendimento automatico applicato nella tecnologia indossabile e nelle sue applicazioni", nel 2017 Conferenza internazionale sui sistemi sostenibili intelligenti (ICISS), pp. 347–354, IEEE , 2017.

[11] M. Bortoli, M. Furini, S. Mirri, M. Montangero e C. Prandi, “Conversational interfaces for a smart campus: A case study,” in Atti della Conferenza Internazionale sulle Interfacce Visive Avanzate, pp 1–5, 2020.

[12] J. Santa e PJ Fernandez, "Connettività ipv6 senza soluzione di continuità per due ruote", Pervasive and Mobile Computing, vol. 42, pagg. 526–541, 2017.

[13] M. Wollschlaeger, T. Sauter e J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0", IEEE industrial electronics magazine, vol. 11, n. 1, pagg. 17–27, 2017.

[14] PP Jayaraman, A. Yavari, D. Georgakopoulos, A. Morshed e A. Za slavsky, "Piattaforma Internet delle cose per l'agricoltura intelligente: esperienze e lezioni apprese", Sensors, vol. 16, n. 11, pag. 1884, 2016.

[15] R. David, J. Duke, A. Jain, VJ Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, et al., “Tensorflow lite micro : apprendimento automatico integrato su sistemi tinyml,” preprint arXiv arXiv:2010.08678, 2020.

[16] B. Sudharsan, JG Breslin e MI Ali, "Edge2train: A framework to training machine learning models (svms) on resource-constrained iot edge device", in Atti della 10a conferenza internazionale sull'Internet delle cose, pp 1–8, 2020.

[17] "Openn." https://www.opennn.net/. Accesso: 08-07-2021.

[18] Dennis, Don Kurian e Gaurkar, Yash e Gopinath, Sridhar e Goyal, Sachin e Gupta, Chirag e Jain, Moksh e Jaiswal, Shikhar e Kumar, Ashish e Kusupati, Aditya e Lovett, Chris e Patil, Shishir G e Saha ., Oindrila e Simhadri, Harsha Vardhan, "EdgeML: Machine Learning per dispositivi edge con risorse limitate".

[19] B. Sudharsan, JG Breslin e MI Ali, "Rce-nn: una pipeline a cinque fasi per eseguire reti neurali (cnns) su dispositivi iot edge con vincoli di risorse", in Atti della 10a conferenza internazionale su Internet delle cose, pp. 1–8, 2020.

[20] C. Banbury, VJ Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau, et al., “Mlperf tiny bench mark”, arXiv preprint arXiv:2106.07597, 2021.

[21] S. Gal-On e M. Levy, "Esplorare coremark un benchmark che massimizza la semplicità e l'efficacia", The Embedded Microprocessor Benchmark Consortium, 2012.

[22] B. Sudharsan, S. Salerno, D.-D. Nguyen, M. Yahya, A. Wahid, P. Yadav, JG Breslin e MI Ali, "Tinyml benchmark: Executing fully connected neural networks on commodity microcontrollers", in IEEE 7th World Forum on Internet of Things (WF-IoT), New Orleans, Louisiana, Stati Uniti, 2021.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., “Scikit- impara: apprendimento automatico in python,” il Journal of machine learning research, vol. 12, pagg. 2825–2830, 2011.

[24] P. Probst, MN Wright e A.-L. Boulesteix, "Iperparametri e strategie di ottimizzazione per la foresta casuale", Recensioni interdisciplinari Wiley: Data Mining e Knowledge Discovery, vol. 9, n. 3, pag. e1301, 2019.

[25] A. Verikas, A. Gelzinis e M. Bacauskiene, "Dati minerari con foreste casuali: un'indagine e risultati di nuovi test", Riconoscimento di modelli, vol. 44, n. 2, pagg. 330–349, 2011.

[26] S. Gonzalez, S. Garc´ya, J. Del Ser, L. Rokach e F. Herrera, “Un tutorial pratico sul bagging e sul potenziamento degli insiemi basati sull'apprendimento automatico: algoritmi, strumenti software, studio delle prestazioni, pratica prospettive e opportunità,” Information Fusion, vol. 64, pp. 205–237, 2020.