UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Accuratezza vs Consumo Energetico: Confronto tra Classical Machine Learning e Tiny Machine Learning

RELATORE

Prof. **Fabio Palomba**

CORRELATORI

Dott. **Vincenzo De Martino**

Dott. **Stefano Lambiase**

Università degli Studi di Salerno

CANDIDATA

Marta Napolillo

Matricola: 0512109836

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

È più facile scendere da una montagna che salirci, ma il panorama è migliore visto dalla cima.

-Michela Murgia

Abstract

Negli ultimi anni, c'è stato un crescente interesse per la Green AI e il Tiny Machine Learning. Tuttavia, mancano articoli che comparino in modo completo i principali framework, PyTorch e TensorFlow, in termini di efficienza energetica e qualità dei modelli. Questa tesi compara PyTorch e TensorFlow utilizzando le Convolutional Neural Networks su due popolari dataset, includendo tecniche di quantizzazione nelle versioni lite dei framework per ridurre le dimensioni dei modelli. I risultati indicano che, nonostante PyTorch consumi più energia, non ci sono differenze significative di accuratezza. Tuttavia, TensorFlow risulta essere più efficiente considerando tutte le metriche definite. Infine, la tecnica di quantizzazione migliore in questo studio è la Float16 Quantization.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

L'ampia diffusione dell'Intelligenza Artificiale in vari contesti ha accentuato la necessità di migliorare l'accuratezza dei modelli, comportando una fase di addestramento più intensiva e dispendiosa a livello di dati e risorse computazionali. Tuttavia, sono emerse significative preoccupazioni riguardo all'impatto ambientale associato a tali processi. Questo ha causato un crescente interesse per la Green AI, che propone soluzioni per contenere l'impatto dei modelli sull'ambiente. In parallelo, è emersa anche una nuova branca del Machine Learning nota come Tiny Machine Learning, con lo scopo di eseguire sui microcontrollori i modelli di dimensioni ridotte, conservando un'alta accuratezza. La Green AI e il Tiny Machine Learning rappresentano due approcci complementari per ridurre l'uso di risorse energetiche e rendere l'Intelligenza Artificiale più sostenibile dal punto di vista ambientale.

1.1 Motivazioni e obiettivi

La seguente tesi ha condotto un confronto tra le Convolutional Neural Network sviluppate per i dataset CIFAR-10 e SVHN mediante l'uso dei framework PyTorch e TensorFlow, ampiamente adottati nell'ambito dello sviluppo di applicazioni di Machine Learning. Lo scopo principale è stato quello di identificare il framework che

ha l'impatto ambientale maggiore, misurando il consumo energetico e le emissioni di CO_2-equivalente prodotte durante le fasi di addestramento e valutazione, e valutare quale dei due framework ha ottenuto le migliori metriche di qualità. In seguito, ai modelli sono state applicate le tecniche di quantizzazione offerte dai framework PyTorch Mobile e TensorFlow Lite. Questa fase è stata condotta con l'obiettivo di valutare quale dei due framework riuscisse a ottimizzare in modo più efficace i modelli. In questo modo, si è valutato quale framework fosse in grado di ridurre significativamente le dimensioni del modello originale, di mantenere al contempo un'alta accuratezza e di garantire un basso consumo energetico durante la fase di valutazione dei modelli quantizzati. La riduzione della dimensione dei modelli è stata ottenuta mediante l'applicazione delle tecniche di quantizzazione dei framework, in quanto vengono ampiamente applicate per tale scopo. Inoltre, questo approccio ha permesso di effettuare un confronto anche tra le versioni "lite", garantendo un adeguato confronto tra i framework più utilizzati nel Machine Learning.

Oltre ai risvolti ambientali, i risultati prodotti in questa tesi permettono una riduzione del costo di sviluppo dei modelli di Machine Learning. Il primo aspetto da considerare è il costo hardware dei microcontrollori. Infatti, questi dispositivi si rivelano notevolmente più economici rispetto alle GPU, spesso impiegate nei data center per l'esecuzione di modelli di Classical Machine Learning. Inoltre, è importante evidenziare la differenza di consumo energetico, in quanto i microcontrollori richiedono una potenza inferiore rispetto alle GPU, che comporta una significativa riduzione dei costi di gestione. In conclusione, l'utilizzo di modelli di Tiny Machine Learning permetto di ottenere un impatto positivo sul bilancio economico di un progetto, rappresentando una svolta importante nei futuri progetti di Machine Learning.

1.2 Risultati ottenuti

I risultati ottenuti in questa ricerca evidenziano che TensorFlow si dimostra più efficiente dal punto di vista energetico e richiede meno tempo sia durante la fase di addestramento che durante la fase di evaluation. Tuttavia, è importante notare che l'accuratezza dei modelli varia notevolmente a seconda del dataset utilizzato. In

particolare, si osserva che PyTorch ottiene risultati migliori con il dataset CIFAR-10, mentre TensorFlow mostra un'accuratezza superiore quando si lavora con SVHN. Per quanto riguarda il Tiny Machine Learning, l'applicazione di tecniche di quantizzazione ha un impatto significativo sull'accuratezza dei modelli. In particolare, la Dynamic Quantization di PyTorch Mobile, la Dynamic Range Quantization e Float16 Quantization di TensorFlow Lite sono le tecniche che conservano meglio l'accuratezza. Invece, la Float16 Quantization di TensorFlow Lite sembra offrire il miglior compromesso tra accuratezza e consumo energetico. In conclusione, per ottenere modelli di Deep Learning con un impatto ambientale ridotto, è preferibile utilizzare TensorFlow durante la fase di addestramento, applicare la Float16 Quantization e eseguire i modelli su microcontrollori. Questo approccio consente di mantenere un equilibrio positivo tra accuratezza e consumo energetico, contribuendo in modo significativo a ridurre l'impatto ambientale associato ai modelli di Machine Learning.

1.3 Struttura della tesi

La struttura della tesi è organizzata come segue:

- **Capitolo 1**: descrive le motivazioni e gli obiettivi alla base dello sviluppo di questa ricerca;
- **Capitolo 2**: analizza gli articoli in merito alla Green AI, i vantaggi e gli svantaggi di sviluppare modelli di Tiny Machine Learning e confronta le tecniche di riduzione della dimensionalità presenti in letteratura;
- **Capitolo 3**: presenta in dettaglio la metodologia della seguente ricerca, differenziando gli step di sviluppo;
- **Capitolo 4**: analizza i diversi risultati ottenuti dalla seguente ricerca, tramite le domande di ricerca;
- **Capitolo 5**: offre una panoramica del progetto e i possibili sviluppi futuri.

CAPITOLO 2

Stato dell'arte

2.1 Green AI

La sempre più ampia adozione dei modelli di Machine Learning in svariati ambiti ha portato all'aumento del consumo energetico richiesto e, di conseguenza, delle emissioni di CO_2-equivalente generate. A tal proposito, i ricercatori hanno incrementato l'interesse per il Green AI [3, 4], con lo scopo di analizzare l'impatto ambientale dei modelli di Intelligenza Artificiale.

Il documento redatto da Verdecchia et al. [4] fornisce una panoramica dei lavori scientifici esistenti relativi alla Green AI. Gli autori hanno notato che i primi articoli sono stati pubblicati a partire dal 2015, con una crescita più o meno costante nel numero di pubblicazioni negli anni successivi. Inoltre, la maggior parte di questi articoli si concentra principalmente sul tema del consumo energetico, come ad esempio l'articolo scritto da Georgiou et al. [3]. In questo articolo, sono stati confrontati i consumi energetici generati da sei modelli differenti in tre ambiti distinti del Deep Learning: Recommender Systems, Natural Language Processing e Computer Vision. I modelli sono stati sviluppati con i framework PyTorch e TensorFlow, al fine di confrontare i differenti costi energetici generati durante la fase di training e di inferenza

dei framework. In generale, i risultati hanno evidenziato che TensorFlow è meno costoso durante la fase di training, mentre PyTorch risulta meno costoso durante la fase di inferenza. Tra i 98 articoli scientifici studiati da Verdecchia et al. [4], solo 8 sono librerie per stimare il consumo energetico e le emissioni di CO_2-equivalente, tra cui è possibile citare eco2AI, la libreria open-source implementata da Budennyy et al. [5]. I ricercatori di eco2AI, al fine di garantire la precisione delle analisi senza sovrastimare i dati generati, hanno effettuato una suddivisione del consumo energetico prodotto dalla GPU, dalla CPU e dalla memoria RAM. Invece, hanno calcolato le emissioni di CO_2-equivalente sul totale del consumo energetico prodotto attraverso la formula 2.1.1, dove γ è il coefficiente di intensità, cioè il peso in chilogrammi di CO_2 emessa per ogni kilowattora (kWh) di elettricità generata da un particolare settore energetico del paese, e PUE è un parametro di default pari ad 1 che indica la potenza usata quando la fase di training è stata effettuata con una piattaforma cloud.

$$CF = \gamma PUE(E_{CPU} + E_{GPU} + E_{RAM}) \tag{2.1.1}$$

Solamente il 23% degli studi presi in analisi da Verdecchia et al. [4] coinvolgono partner industriali. Tuttavia, i risultati mettono in evidenza che è fondamentale riconoscere l'impatto significativo della Green AI, dimostrando che il beneficio in termini di riduzione del consumo energetico si attesta intorno al 115%, il tutto a un costo minimo o addirittura nullo in termini di perdita di accuratezza.

2.2 Introduzione al Tiny Machine Learning

Nel 2014, il team responsabile dello sviluppo di OK Google, ha raggiunto un notevole traguardo nel campo dell'Intelligenza Artificiale sviluppando un modello di soli 14 kilobyte (KB) al fine di renderlo disponibile su un microcontrollore. Come descritto nell'introduzione del libro *TinyML - Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers* [6] scritto da Warden e Situnayake, questo risultato ha aperto la strada a un nuovo campo del Machine Learning, noto come Tiny Machine Learning[1]. Gli autori spiegano che l'obiettivo principale del

[1] https://www.tinyml.org/

Tiny Machine Learning è di ridurre le dimensioni dei modelli in modo significativo per eseguirli sui microcontrollori (MCUs), senza influenzarne troppo l'accuratezza. Inoltre, è strettamente correlato al concetto di Green AI, in quanto l'esecuzione sui microcontrollori permette di ridurre il consumo energetico associato all'intero ciclo di vita dei modelli. Nonostante il Tiny Machine Learning ha molti vantaggi economici e ambientali, presenta numerosi limiti che non ne permettono il suo utilizzo effettivo. In seguito, sono evidenziate le principali differenze hardware e software tra i modelli di Classical Machine Learning e Tiny Machine Learning.

2.2.1 Hardware

Il Classical Machine Learning, integrato con i dispositivi IoT, utilizza un'infrastruttura cloud per trasferire e ricevere dati dal server. In questa architettura, il Tiny Machine Learning può essere un livello aggiuntivo per il filtraggio di alcune operazioni, permettendo una gestione ottimale dei componenti dell'architettura riducendo il carico di lavoro dei server e delle comunicazioni nella rete [1]. Le principali differenze hardware sono riassunte nella Tabella 2.1, la quale evidenzia le risorse disponibili con la conseguente riduzione del costo dell'hardware e della potenza richiesta, e di conseguenza anche del consumo energetico necessario. Invece, il Mobile Machine Learning è un ibrido tra il Classical Machine Learning e il Tiny Machine Learning, che permette di eseguire i modelli su dispositivi mobile, come smartphone e tablet.

Piattaforma	Architettura	RAM	Memoria	Potenza	Prezzo
Classical ML	GPU	HBM	SSD/Disk		
Nvidia V100	Nvidia Volta	16GB	TB~PB	250W	$9K
Mobile ML	CPU	DRAM	Flash		
Cell Phone	Mobile CPU	4GB	64GB	~8W	~$750
Tiny ML	MCU	SRAM	eFlash		
IoT devices	Arm	~320KB	~1MB	~0.3W	~$5

Tabella 2.1: Confronto tra l'hardware del Classical ML, Mobile ML e Tiny ML [2].

2.2.2 Software

Il principale svantaggio del Tiny Machine Learning a livello software è rappresentato dalla mancanza, sui microcontrollori, di un Sistema Operativo ricco di risorse, causando un aumento del costo di sviluppo del software [1]. Infatti, i framework più utilizzati per il Classical Machine Learning, come PyTorch[2] e TensorFlow[3], non sono adatti per lo sviluppo di un modello di Tiny Machine Leaning. Dall'altro lato, esistono tre modalità per lo sviluppo del Tiny Machine Learning:

- **Manual Programming**: Con lo scopo di distribuire manualmente i modelli sui microcontrollori, ottenendo sempre la soluzione migliore. Ma l'eterogeneità dei microcontrollori disponibili ne ostacola la diffusione, dato che il costo di sviluppo è molto elevato [1].

- **Code Generation**: Crea il modello direttamente utilizzando il linguaggio C. Questo approccio consente di aumentare l'accuratezza, ridurre le dimensioni del modello, ma comporta l'inconveniente che il codice generato non è portabile su altri sistemi [1, 2].

- **Interpreter**: Prevede di creare un modello con i framework PyTorch o TensorFlow, di ridurre la dimensione del modello attraverso le versioni "lite" dei framework (rispettivamente PyTorch Mobile[4] e TensorFlow Lite[5]), e ottenere il modello tiny. Questo metodo ha minori costi di sviluppo, a svantaggio, però, dell'accuratezza e della dimensione del modello [1, 2].

Il Tiny Machine Learning permette di eseguire l'inferenza del modello nei dispositivi IoT, determinando i seguenti vantaggi software [1, 7]:

- una riduzione della latenza, in quanto i dati sono elaborati in locale;

- una riduzione delle comunicazioni sulla rete, con una conseguente diminuzione del carico di lavoro dei server ed un aumento della sicurezza dei dati.

[2] https://pytorch.org/
[3] https://www.tensorflow.org/
[4] https://pytorch.org/mobile/home/
[5] https://www.tensorflow.org/lite/

2.2.3 Sviluppo di un modello di Tiny Machine Learning

Per sviluppare un modello di Tiny Machine Learning, spesso si utilizza un modello creato tramite il Classical Machine Learning [7]. Infatti, come evidenziato dalla Figura 2.1, il processo per implementare un modello di Tiny Machine Learning utilizzando l'approccio Interpreter [1] si divide in tre fasi sequenziali:

1. **Training**: Si utilizzano i framework per il Classical Machine Learning, come TensorFlow e PyTorch, per creare ed addestrare il modello sul dataset.

2. **Optimizing**: Vengono applicate le tecniche per ridurre la dimensione del modello, come la quantizzazione durante o post-training e il pruning.

3. **Deploying**: Il modello viene eseguito sui microcontrollori per il calcolo dell'inferenza sui dati prodotti dai sensori.

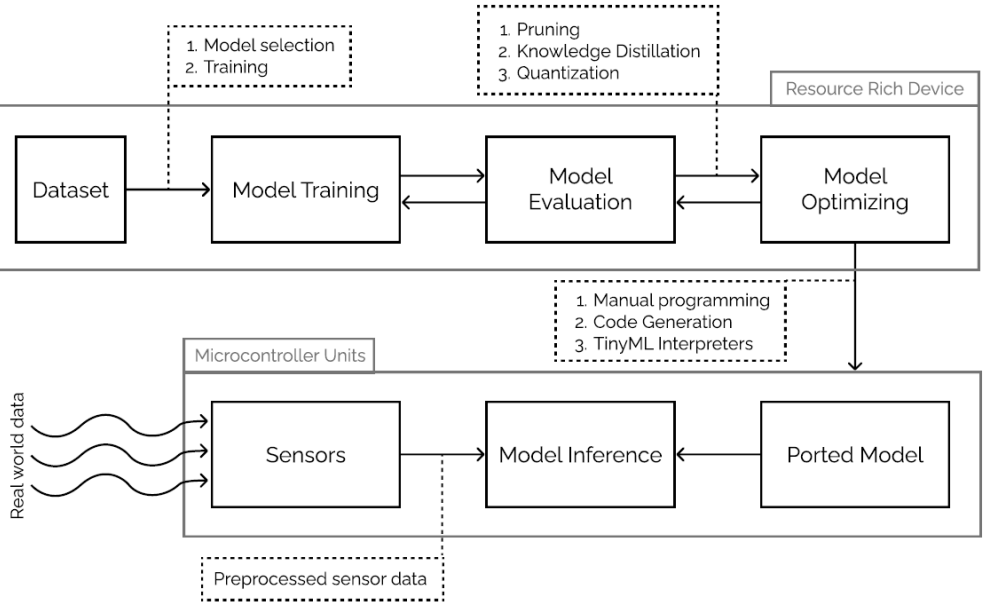

Figura 2.1: Pipeline per lo sviluppo del Tiny Machine Learning [1].

2.3 Riduzione della dimensionalità dei modelli

Essendo il Tiny ML una tecnologia emergente, esistono pochi progetti che evidenziano il suo vantaggio. Un esempio di progetto che ha effettuato un confronto sui diversi campi del Machine Learning è MicroNets [2]. Infatti, Banbury et al. hanno effettuato un'ottimizzazione, attraverso il framework TensorFlow Lite, delle reti neurali per Anomaly Detection, Image Classification, Keyword Spotting, Noise Suppression, Object Detection, Speech Recognition, Superresolution e Visual Wake Words. I metodi di ottimizzazione che hanno utilizzato sono basati sulla tecnica Differentiable Neural Architecture Search (DNAS), cioè basati sul gradiente per la ricerca di architetture di rete neurale ottimali. Dimostrando così che è possibile costruire automaticamente modelli che si adattino ai vincoli dei microcontrollori, massimizzando le prestazioni e l'accuratezza. Di conseguenza, hanno affermato che il vantaggio energetico dovrebbe motivare lo sviluppo di modelli di Tiny Machine Learning.

D'altra parte, esistono articoli scientifici che affrontano la riduzione della dimensione dei modelli attraverso la Post-Training Quantization (PTQ) per utilizzare il modello creato tramite il Classical Machine Learning e in seguito effettuare la quantizzazione. Ad esempio, Sailesh et al. [8] hanno sviluppato un framework open-source per eseguire la quantizzazione post-training su modelli di Convolutional Neural Networks (CNN) e renderli disponibili su microcontrollori basati su ARM Cortex M. Il framework, per effettuare la quantizzazione dei pesi e dei bias da float32 a int8, prevede l'utilizzo di modelli già creati e addestrati in TensorFlow. Infine, il modello risultante è reso disponibile in linguaggio C per effettuare l'inferenza e, successivamente, essere eseguito direttamente sul microcontrollore.

In generale, lo svantaggio principale della Post-Training Quantization è la perdita di accuratezza. A tal fine, è stata proposta la Quantization-Aware Training (QAT) che permette di effettuare la quantizzazione durante la fase di training. Lo studio di Ghamari et al. [9] presenta la Quantization-Guided Training (QGT) per i modelli di reti neurali, che permette di adattare la funzione di perdita durante l'allenamento evitando gli errori di quantizzazione. I risultati mostrano che tale tecnica, oltre a

ridurre notevolmente la dimensione del modello, permette di evidenziare i colli di bottiglia, ovvero strati che ostacolano la quantizzazione.

Una delle più importanti tecniche di riduzione della memoria, oltre alla quantizzazione, è il pruning. Esso permette di rimuovere interi livelli della rete neurale finché non si raggiunge la dimensione desiderata. De Leon e Atienza [10] propongono la tecnica di Depth Pruning che permette di ridurre i requisiti computazionali dei modelli di rete neurale e non richiede hardware specializzato. Per migliorare questo aspetto, gli autori propongono di utilizzare una rete ausiliaria altamente efficiente che lavora da interprete intermedio durante l'esecuzione. Questa soluzione permette di non avere significative variazioni per l'accuratezza, consentendo di ridurre ugualmente la dimensione del modello.

Considerati i rilevanti risultati provenienti dagli articoli disponibili, l'obiettivo centrale della ricerca svolta in questa tesi è porre particolare attenzione sul trade-off tra le metriche di qualità e l'impatto ambientale dei modelli. Infatti, le finalità del progetto includono: lo sviluppo del medesimo modello per effettuare un confronto energetico tra i framework PyTorch e TensorFlow, la valutazione delle varie tecniche di compressione della memoria offerte da PyTorch Mobile e TensorFlow Lite e, infine, lo studio dell'influenza che le tecniche esercitano sulle diverse metriche di qualità.

CAPITOLO 3

Metodologia di ricerca

3.1 Presentazione del progetto

La ricerca in questione ha coinvolto la creazione di due diverse architetture per la classificazione di immagini, una per il dataset CIFAR-10 e una per il dataset SVHN. L'obiettivo è condurre una comparazione, attraverso metriche di qualità e metriche per analizzare l'impatto ambientale, tra i modelli generati tramite i framework PyTorch[1] e TensorFlow[2]. Dopo aver completato le fasi di training ed evaluation dei modelli di Classical Machine Learning, sono state utilizzate le tecniche di quantizzazione disponibili in PyTorch Mobile[3] e TensorFlow Lite[4], con lo scopo di individuare la miglior tecnica di riduzione della dimensionalità. Per concludere, è stata eseguita la fase di evaluation dei modelli di Tiny Machine Learning al fine di analizzare eventuali riduzioni nell'accuratezza ottenuta. Per chiarire al meglio i risultati ottenuti, sono state formulate le seguenti tre domande di ricerca.

[1] https://pytorch.org/
[2] https://www.tensorflow.org/
[3] https://pytorch.org/mobile/home/
[4] https://www.tensorflow.org/lite/

RQ$_1$. *Quale dei due framework è più costoso dal punto di vista ambientale ed energetico in fase di training ed evaluation?*

La prima domanda di ricerca mira a determinare quale dei due framework, PyTorch e TensorFlow, abbia l'impatto ambientale più significativo. I risultati sono stati prodotti nel corso del training e dell'evaluation dei modelli di Classical Machine Learning, misurando il consumo energetico prodotto e le emissioni di CO_2-equivalente attraverso la libreria CodeCarbon[5].

RQ$_2$. *Quale modello di Machine Learning ha caratteristiche di qualità migliori?*

La seconda domanda di ricerca consente di identificare il framework ottimale per i modelli analizzati, basandosi sulle metriche di accuracy, precision, recall, F1-score, dimensioni e numero di parametri dei modelli. I risultati prodotti nella RQ1 e RQ2 permettono di esaminare il trade-off tra accuratezza e consumo energetico, al fine di determinare nel complessivo il framework ideale per la creazione di modelli di Green Machine Learning.

RQ$_3$. *Quale tecnica effettua un'ottimizzazione migliore dei modelli?*

La terza domanda di ricerca ha l'obiettivo di identificare la tecnica di quantizzazione più efficace per i framework PyTorch Mobile e TensorFlow Lite. Le metriche considerate in questo contesto comprendono accuracy, precision, recall, F1-score, dimensioni dei modelli, consumo energetico ed emissioni di CO_2-equivalente generate durante la fase di evaluation. Inoltre, per ciascun framework, sarà individuata la tecnica di quantizzazione ottimale, e in conclusione verrà identificata la migliore tra tutte le tecniche analizzate.

3.2 Selezione dei dataset e definizione dell'architettura

Nel corso del progetto, sono state eseguite diverse fasi consecutive per garantire risultati ottimali nella classificazione di immagini. La prima di queste è stata la

[5] `https://codecarbon.io/`

selezione accurata dei dataset disponibili adatti alla classificazione di immagini.
Al fine di confrontare dataset di dimensione differenti, ma entrambi di tipologia multiclasse e con immagini a colori, i dataset selezionati sono:

- **CIFAR-10**[6]: Comprende dieci diverse categorie di immagini che rappresentano animali e veicoli. Di default, il dataset è suddiviso in due parti principali: un insieme di dati di training con 50.000 campioni (5.000 campioni per classe) e un insieme di dati di test con 10.000 campioni (1.000 campioni per classe), in modo da garantire una distribuzione bilanciata delle classi all'interno del dataset.

- **SVHN**[7]: É derivato dai numeri civici presenti nelle immagini di Google Street View e contiene cifre numeriche da 0 a 9. SVHN è suddiviso principalmente in tre parti: un set di dati di allenamento con 73.257 campioni, un set di dati di test con 26.032 campioni e un set aggiuntivo di 13.068 campioni. È importante notare che SVHN non presenta una distribuzione bilanciata tra le cifre rappresentate.

Dopo aver selezionato i dataset da utilizzare, è stata eseguita una fase di normalizzazione dei valori dei pixel delle immagini. Poiché entrambi i dataset contengono immagini a colori, questa normalizzazione è stata applicata a ciascuno dei tre canali (red, green e blue) con l'obiettivo di impostare sia la media che la deviazione standard a 0.5 per ciascun canale. Questa operazione di ridimensionamento è stata eseguita per assicurare che i valori dei pixel rientrassero nell'intervallo compreso tra -1 e 1, garantendo così una scala uniforme su tutti i canali.

La tipologia di rete neurale selezionata è la Convolutional Neural Network (CNN), poiché è stata appositamente progettata per emulare il sistema visivo umano e migliorare l'analisi delle immagini. Nei modelli di CNN, i layer fondamentali sono:

- **Convolutional Layer**: Questo strato applica filtri di convoluzione alle immagini per rilevare caratteristiche come bordi, curve e forme, consentendo di estrarre automaticamente le caratteristiche più rilevanti dalle immagini, indipendentemente dalla loro posizione o dimensione all'interno dell'immagine.

[6] https://www.cs.toronto.edu/~kriz/cifar.html
[7] http://ufldl.stanford.edu/housenumbers/

- **Pooling Layer**: Solitamente collocato successivamente ai livelli di convoluzione, il pooling layer svolge il ruolo di ridurre le dimensioni delle mappe delle caratteristiche. Questa riduzione avviene attraverso operazioni di pooling comuni come il max-pooling e l'average-pooling.

- **Fully Connected Layer**: In questo livello ogni neurone riceve input da tutti i neuroni dello strato precedente. É spesso utilizzato per fare previsioni finali o decisioni basate sulle caratteristiche estratte.

- **Activation Function**: La funzione di attivazione introduce una non-linearità ai dati, permettendo al modello di apprendere rappresentazioni complesse e di risolvere problemi non lineari.

Poiché i dataset sono ampiamente utilizzati, entrambe le architetture sono state reperite dalla piattaforma Kaggle[8]. In particolare, l'architettura utilizzata per SVHN[9] non ha subito alcuna modifica, mentre quella impiegata per il dataset CIFAR-10[10] ha subito una variazione dei layer BatchNorm2d con i layer Dropout. Tale modifica ha permesso di prevenire l'overfitting causando l'aumento dell'accuratezza [11]. Nella Tabella 3.1 è illustrata l'architettura della CNN per il dataset CIFAR-10, mentre nella Tabella 3.2 è presentata l'architettura per il dataset SVHN. Le dimensioni di input e output di ciascun livello sono espresse utilizzando la notazione "Canali x Altezza x Larghezza" per rappresentare le dimensioni dei tensori nelle diverse fasi.

[8] https://www.kaggle.com/

[9] https://www.kaggle.com/code/dimitriosroussis/svhn-classification-with-cnn-keras-96-acc

[10] https://www.kaggle.com/code/abhishekshaw21/cifar-10-using-pytorch

Operatore	Dimensione Input	Dimensione Output
Conv2d - ReLU	3 x 32 x 32	32 x 32 x 32
Conv2d - ReLU	32 x 32 x 32	64 x 32 x 32
MaxPool2d	64 x 32 x 32	64 x 16 x 16
Dropout	64 x 16 x 16	64 x 16 x 16
Conv2d - ReLU	64 x 16 x 16	128 x 16 x 16
Conv2d - ReLU	128 x 16 x 16	128 x 16 x 16
MaxPool2d	128 x 16 x 16	128 x 8 x 8
Dropout	128 x 8 x 8	128 x 8 x 8
Conv2d - ReLU	128 x 8 x 8	256 x 8 x 8
Conv2d - ReLU	256 x 8 x 8	256 x 8 x 8
MaxPool2d	256 x 8 x 8	256 x 4 x 4
Dropout	256 x 4 x 4	256 x 4 x 4
Flatten	256 x 4 x 4	4096
Linear - ReLU	4096	1024
Linear - ReLU	1024	512
Linear	512	10

Tabella 3.1: Tabella riassuntiva dell'architettura creata per il dataset CIFAR-10

Operatore	Dimensione Input	Dimensione Output
Conv2d - ReLU	3 x 32 x 32	32 x 32 x 32
BatchNorm2d	32 x 32 x 32	32 x 32 x 32
Conv2d - ReLU	32 x 32 x 32	32 x 32 x 32
MaxPool2d - Dropout	32 x 32 x 32	32 x 16 x 16
Conv2d - ReLU	32 x 16 x 16	64 x 16 x 16
BatchNorm2d	64 x 16 x 16	64 x 16 x 16
Conv2d - ReLU	64 x 16 x 16	64 x 16 x 16
MaxPool2d - Dropout	64 x 16 x 16	64 x 8 x 8
Conv2d - ReLU	64 x 8 x 8	128 x 8 x 8
BatchNorm2d	128 x 8 x 8	128 x 8 x 8
Conv2d - ReLU	128 x 8 x 8	128 x 8 x 8
MaxPool2d - Dropout	128 x 8 x 8	128 x 4 x 4
Flatten	128 x 4 x 4	2048
Linear - ReLU - Dropout	2048	128
Linear	128	10

Tabella 3.2: Tabella riassuntiva dell'architettura creata per il dataset SVHN

3.3 Training ed evaluation

Prima di avviare il processo di training, è fondamentale definire gli Hyper-Parameters, in quanto regolano il comportamento del modello e possono influire in modo sostanziale sulle sue prestazioni. Durante questa fase, i valori degli Hyper-Parameters sono stati configurati manualmente attraverso un processo iterativo al fine di individuare la combinazione migliore. La Tabella 3.3 riassume gli Hyper-Parameters scelti in base al dataset, e includono:

- **Learning Rate**: Questo parametro regola l'entità con cui i pesi della rete vengono aggiornati in risposta all'errore calcolato. Aumentare il tasso di apprendimento può accelerare il processo di addestramento, a svantaggio delle prestazioni.
- **Number of epochs**: Ogni iterazione sull'intero dataset è chiamata epoca. Un aumento del numero di epoche può portare a una migliore apprendimento dai dati, tuttavia comporta anche il rischio di overfitting.
- **Batch Size**: La scelta della dimensione del batch determina quanti campioni di dati vengono elaborati simultaneamente durante ogni iterazione, influenzando sia l'efficienza del processo di training che la quantità di memoria necessaria.

Dataset	Learning Rate	Number of epochs	Batch Size
CIFAR-10	0,001	100	64
SVHN	0,001	100	128

Tabella 3.3: Hyper-Parameters utilizzati per le diverse architetture

In fase di training, è stato utilizzato l'ottimizzatore Adam [12], noto per la sua capacità di adattare automaticamente il tasso di apprendimento. Invece, la funzione di perdita adottata è la cross-entropy loss [13], una scelta comune per problemi di classificazione di immagini. Le fasi di training ed evaluation sono state effettuate nell'ambiente di servizio cloud chiamato Colaboratory[11], al fine di poter utilizzare l'hardware disponibili su tale piattaforma.

[11] https://colab.research.google.com/

Per il training e l'evaluation dei modelli di Classical Machine Learning è stata la utilizzata GPU Tesla T4® basata sull'architettura NVIDIA-SMI contenente 40 core, con 12.7 GB RAM di sistema, 15 GB di RAM della GPU e 78.2 GB di HDD. Invece, per la fase di evaluation dei modelli di Tiny Machine Learning è stata utilizzata la CPU Intel® Xeon® con 2.20 GHz, 12.7 GB di RAM e 107.7 GB di HDD.

3.4 Metriche energetiche e di qualità

Dopo aver completato la fase di addestramento dei modelli di Classical Machine Learning, si è proceduto a salvarli in modo da poter applicare in seguito le tecniche di quantizzazione e ottenere così modelli "tiny". Successivamente, è stata condotta una fase di valutazione sia per i modelli di Classical Machine Learning sia per i modelli di Tiny Machine Learning ottenuti. Durante questa fase di valutazione, sono state calcolate diverse metriche di qualità, comprese la F1-score, l'accuracy, la precision e il recall, utilizzando la libreria scikit-learn. Poiché i problemi di classificazione analizzati implicano una suddivisione in più classi, si è scelto di utilizzare la tecnica di macro-averaging per ottenere un singolo valore per ciascuna metrica.

Inoltre, sia durante la fase di addestramento che durante la fase di valutazione, sono stati monitorati tre parametri importanti: le emissioni di CO_2-equivalente, il consumo energetico e il tempo di esecuzione. Questo monitoraggio è stato effettuato utilizzando la libreria CodeCarbon[12]. In particolare, l'uso del decoratore `@track_emissions` ha permesso di registrare tutte le informazioni sull'esecuzione in un file CSV dedicato, fornendo così una visione dettagliata dell'impatto ambientale e delle risorse impiegate nei processi di addestramento e valutazione dei modelli. Per i modelli di Machine Learning tradizionali, sono state effettuate anche ulteriori misurazioni, tra cui il numero di parametri del modello. Questa informazione è stata ottenuta utilizzando il metodo `count_params()` della libreria TensorFlow, invece per i modelli basati su PyTorch è stata utilizzata la funzione `get_model_complexity_info()` dalla libreria ptflops[13]. Infine, per tutti i modelli è stata calcolata la loro dimensione.

[12] `https://codecarbon.io/`

[13] `https://pypi.org/project/ptflops/`

Questo si è ottenuto salvando temporaneamente i modelli e misurando la dimensione dei file risultanti. Tale procedura fornisce una stima della quantità di spazio di archiviazione necessario per ospitare i modelli.

3.5 Applicazione delle tecniche di quantizzazione

La quantizzazione è una delle tecniche più comuni per ridurre la dimensione dei modelli di Machine Learning su dispositivi con risorse limitate. Essa comprende una serie di metodi che mirano a diminuire la precisione dei calcoli effettuati effettuando un'approssimazione da valori continui a valori discreti. Dalla revisione dello stato dell'arte, è emerso che esistono due approcci principali alla quantizzazione: Quantization Aware Training e Post Training Quantization. Nel contesto del progetto, sono state applicate esclusivamente le tecniche di Post Training Quantization fornite dai framework TensorFlow Lite e PyTorch Mobile, di seguito elencate.

3.5.1 TensorFlow Lite

Dynamic range quantization[14] Tale tecnica è una delle prime da considerare quando si valutano metodi di quantizzazione per reti neurali profonde, in quanto può essere combinata con altre ottimizzazioni per ridurre il consumo di risorse. Durante la conversione, i pesi vengono quantizzati staticamente da valori continui a valori discreti ad 8 bit, mentre le attivazioni, ovvero gli output intermedi del modello durante l'inferenza, vengono quantizzate dinamicamente.

Full integer quantization[15] Per ottenere una riduzione ancora maggiore della memoria e per rendere il modello disponibile su vari microcontrollori, è possibile estendere l'applicazione della quantizzazione intera all'intero modello. È essenziale utilizzare un dataset rappresentativo, come nel nostro caso con 100 elementi. Tuttavia, è importate valutare attentamente le implicazioni sulla precisione del modello,

[14] https://www.tensorflow.org/lite/performance/post_training_quantization#dynamic_range_quantization

[15] https://www.tensorflow.org/lite/performance/post_training_quantization#full_integer_quantization

assicurandosi che il modello quantizzato soddisfi ancora i requisiti di accuratezza dell'applicazione prima di distribuirlo sui microcontrollori.

Float16 quantization[16] Un'altra tecnica che consente di ridurre la dimensione del modello con una perdita minima di accuratezza è la quantizzazione a float16. Questa tecnica permette di dimezzare le dimensioni del modello, in quanto tutti i pesi vengono rappresentati con la metà dei bit rispetto alle loro dimensioni originali. Tuttavia, è importante notare che il principale svantaggio di questa tecnica è la minima riduzione della latenza rispetto ad altre tecniche.

3.5.2 PyTorch Mobile

Static Quantization[17] La quantizzazione statica post-training è una tecnica molto efficace per ridurre la dimensione dei modelli di Convolutional Neural Network. Questa tecnica, dopo la fase di training, coinvolge la conversione delle attivazioni e dei pesi del modello in rappresentazioni di interi a 8 bit.

Dynamic Quantization[18] A differenza della quantizzazione statica, la seguente tecnica non è molto consigliata per i ridurre la dimensione dei modelli di Convolutional Neural Network. Infatti, tra gli operatori disponibili, nelle architetture definite è possibile quantizzare solo i livelli che utilizzano l'operatore Linear.

[16] https://www.tensorflow.org/lite/performance/post_training_quantization#float16_quantization

[17] https://pytorch.org/docs/stable/quantization.html#post-training-static-quantization

[18] https://pytorch.org/docs/stable/quantization.html#post-training-dynamic-quantization

CAPITOLO 4

Analisi dei risultati

Nel seguente capitolo sono stati riportati e analizzati i risultati ottenuti, al fine di risponde delle domande di ricerca discusse nella metodologia.

4.1 RQ1: Impatto ambientale in fase di training ed evaluation per il Classical Machine Learning

RQ$_1$. *Quale dei due framework è più costoso dal punto di vista ambientale ed energetico in fase di training ed evaluation?*

La Tabella 4.1 illustra le differenze energetiche tra i due framework per i risultati prodotti dall'architettura utilizza per il dataset CIFAR-10. Sia durante la fase di training che durante la fase di evaluation, i risultati sono costantemente superiori quando si utilizza il framework PyTorch. Dunque, in questo caso dal punto di vista energetico TensorFlow ottiene risultati migliori.

Framework	Fase	Durata	Emissioni di CO_2	Consumo energetico
PyTorch	Training	28.69 min	24.85 g	54.91 Wh
	Evaluation	0.138 min	0.0312 g	0.19 Wh
TensorFlow	Training	22.63 min	15.28 g	43.75 Wh
	Evaluation	0.019 min	0.0107 g	0.0306 Wh

Tabella 4.1: Risultati prodotti dalla libreria CodeCarbon per il dataset CIFAR-10

Invece, la Tabella 4.2 presenta i risultati ottenuti dai vari framework nel contesto del dataset SVHN. È evidente che anche in questo caso TensorFlow, sia durante la fase di addestramento che durante quella di evaluation, mostra un impatto ambientale inferiore rispetto a PyTorch.

Framework	Fase	Durata	Emissioni di CO_2	Consumo energetico
PyTorch	Training	187.190 min	72.29 g	253.19 Wh
	Evaluation	0.138 min	0.0542 g	0.19 Wh
TensorFlow	Training	91.654 min	57.62 g	127.31 Wh
	Evaluation	0.053 min	0.0353 g	0.0781 Wh

Tabella 4.2: Risultati prodotti dalla libreria CodeCarbon per il dataset SVHN

Diversamente dall'articolo di Georgiou et al. [3], che ha concluso che TensorFlow risulta meno costoso durante la fase di training, mentre PyTorch risulta più vantaggioso durante la fase di evaluation, la seguente ricerca conferma TensorFlow come la scelta ottimale in entrambe le fasi grazie alla sua efficienza energetica e alle prestazioni complessive dei modelli ottenuti. Tali risultati potrebbero derivare anche dalla minore durata richiesta da TensorFlow sia durante la fase di training che in quella di evaluation, rispetto a PyTorch.

4.2 RQ2: Analisi delle metriche di qualità per i modelli di Classical Machine Learning

RQ$_2$. *Quale modello di Machine Learning ha caratteristiche di qualità migliori?*

Per rispondere a questa domanda di ricerca, iniziamo con la presentazione dei risultati ottenuti dai modelli, distinguendo per architettura. Successivamente, procediamo con un'analisi complessiva dei risultati ottenuti.

4.2.1 Analisi dei risultati prodotti per il dataset CIFAR-10

Il modello prodotto del framework PyTorch sull'architettura del dataset CIFAR-10 ha una dimensione di **23.41 MB** e un totale di 5.850.000 parametri. Inoltre, i risultati in fase di evaluation mostrano l'**84.29%** di accuracy, l'84.56% di precision, l'84.29% di recall e l'84.24% di F1-score. La Figura 4.1 illustra la matrice di confusione tra i valori reali e quelli predetti durante la fase di evaluation. Complessivamente il modello ha ottenuto risultati buoni nella classificazione, ma sono state osservate confusioni significative tra le classi 'cat' e 'dog', e in misura minore tra la classe 'automobile' e la classe 'truck'. Inoltre, si è osservato che la classe 'bird' è stata spesso confusa con le altre classi che identificano gli animali, soprattutto con la classe 'dog'.

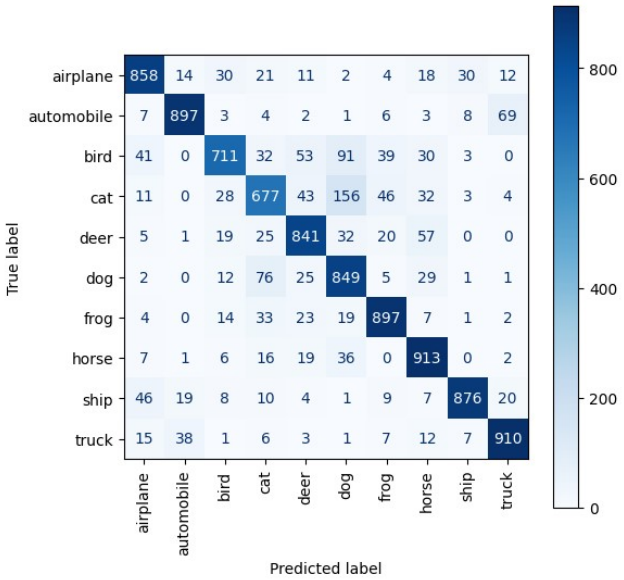

Figura 4.1: Matrice di confusione per il framework PyTorch e il dataset CIFAR-10

Il framework TensorFlow, utilizzato con il dataset CIFAR-10, ha generato un modello di dimensioni pari a **70.29 MB**, contenente 5.851.338 parametri. I risultati in fase di evaluation evidenziano l'**80.89%** di accuracy, l'81.32% di precision, l'80.89% di recall e l'80.79% di F1-score. In generale, come illustrato dalla matrice di confusione nella Figura 4.2, il modello effettua una buona classificazione. Tuttavia, il modello commette degli errori nel classificare alcuni campioni della classe 'cat' come appartenenti alla classe 'dog'. Inoltre, si può notare che le classi che identificano gli animali presentano una classificazione meno accurata rispetto alle altre classi.

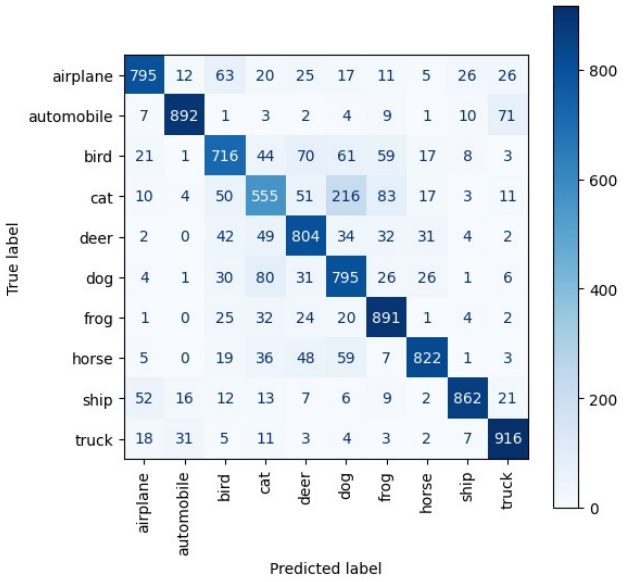

Figura 4.2: Matrice di confusione per il framework TensorFlow e il dataset CIFAR-10

L'architettura per il dataset CIFAR-10 dimostra una maggiore accuratezza quando implementata nel framework PyTorch, evidenziando una classificazione più precisa dei dati nelle matrici di confusione. Infatti, gli errori commessi in PyTorch risultano essere più elevati quando si utilizza il framework TensorFlow. É interessante notare che, nonostante la dimensione del modello PyTorch sia molto inferiore rispetto a quella del modello TensorFlow, entrambi hanno un numero simile di parametri.

4.2.2 Analisi dei risultati prodotti per il dataset SVHN

L'architettura utilizzata per il dataset SVHN con il framework PyTorch ha prodotto un modello con l'**87.03%** di accuracy, l'85.45% di precision, l'84.08% di recall e l'83.04%

di F1-score. Questo modello ha una dimensione di **2.21 MB** e contiene 551.020.000 parametri. Tuttavia, è importante notare che il modello ha mostrato difficoltà nella corretta classificazione del numero 9, confondendolo quasi sempre con il numero 6, come si può notare dalla matrice di confusione presente della Figura 4.3. Questo problema potrebbe derivare da diversi fattori, tra cui un possibile sbilanciamento nel dataset, con un numero leggermente superiore di istanze di 6 rispetto a 9. Inoltre, la somiglianza nella conformazione visuale tra il numero 6 e il numero 9 potrebbe avere influito sulla difficoltà di classificazione, rendendo la problematica legata all'architettura della rete neurale utilizzata.

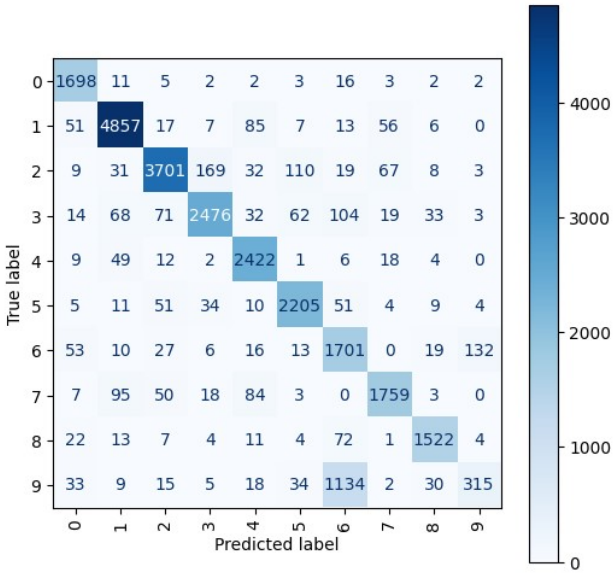

Figura 4.3: Matrice di confusione per il framework PyTorch e il dataset SVHN

Il modello creato utilizzando il framework TensorFlow per il dataset SVHN ha una dimensione di **6.72 MB** e contiene 551,500 parametri. Questo modello ha ottenuto risultati eccellenti nella classificazione con il **95.97%** di accuracy, il 95.49% di precision, il 95.68% di recall e il 95.57% di F1-score. Infatti, la matrice di confusione nella Figura 4.2 non presenta evidenti errori di classificazione.

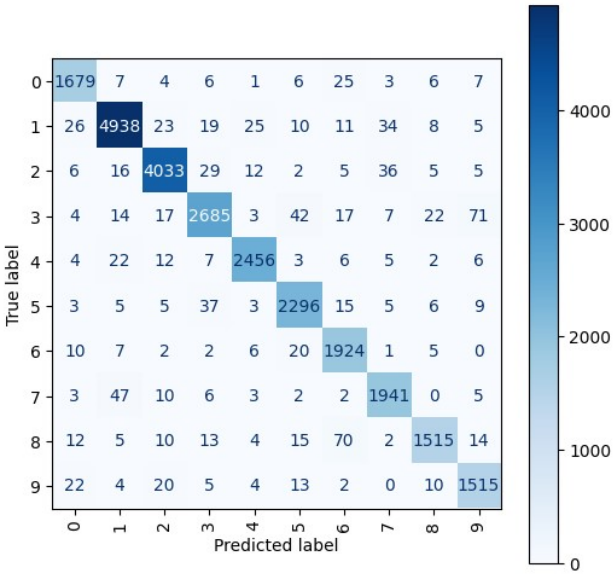

Figura 4.4: Matrice di confusione per il framework TensorFlow e il dataset SVHN

Per il dataset SVHN, il framework che raggiunge una maggiore accuratezza è TensorFlow in quanto effettua provisioni molto accurate in tuute le classi. Ma si può notare che il modello prodotto con PyTorch richiede minor spazio di memoria rispetto a TensorFlow. Nel caso del dataset SVHN, TensorFlow raggiunge una maggiore accuratezza, con previsioni molto precise in tutte le classi. Tuttavia, è importante notare che il modello addestrato con PyTorch richiede meno spazio di memoria rispetto a TensorFlow.

Considerando anche i risultati prodotti con CIFAR-10, è interessante notare che PyTorch si comporta meglio con il dataset CIFAR-10, mentre TensorFlow ha prestazioni superiori con SVHN. Questi risultati indicano che non esiste un framework migliore, in quanto non è possibile stabilire una significativa differenza in termini di accuratezza tra i due framework. In definitiva, sia TensorFlow che PyTorch sono eccellenti strumenti per il deep learning, e la scelta tra di essi in termini di accuratezza è strettamente dipendente dall'architettura del modello utilizzata e dalle specifiche esigenze del progetto. Considerando anche i risultati mostrati nella RQ1, si può osservare che in entrambi i casi, sia con l'architettura per CIFAR-10 sia per SVHN, la fase di training ed evaluation dei modelli richiede meno tempo con PyTorch rispetto a TensorFlow. Tuttavia, è importante notare che i modelli addestrati con TensorFlow tendono a

occupare più spazio di memoria rispetto a quelli addestrati con PyTorch. Questa differenza può essere attribuita alle diverse modalità di gestione delle risorse durante la fase di addestramento dei modelli nei due framework. Quindi, per ottenere un minor impatto ambientale è preferibile utilizzare il framework TensorFlow.

4.3 RQ3: Confronto tra i risultati prodotti dai modelli di Tiny Machine Learning

RQ$_3$. *Quale tecnica effettua un'ottimizzazione migliore dei modelli?*

Al fine di rispondere a tale domanda, sono stati inizialmente esaminati i risultati ottenuti applicando le tecniche di quantizzazione ai modelli differenziando l'architettura e, in seguito, sono stati analizzati complessivamente i risultati ottenuti.

4.3.1 Analisi dei risultati prodotti per il dataset CIFAR-10

La Tabella 4.3 mostra la dimensione dei modelli e le metriche di accuratezza ottenute dopo l'applicazione delle quantizzazioni del framework PyTorch Mobile. La tecnica di Static Quantization ha ridotto la dimensione del modello del 75.03%, mentre la Dynamic Quantization l'ha ridotta del 60.56%. Tuttavia, è importante notare che la Static Quantization ha comportato una significativa diminuzione anche dell'accuratezza del modello, pari al 58.26%, mentre la Dynamic Quantization ha avuto un impatto molto meno significativo, riducendo l'accuratezza solo dello 0.04%.

Quantizzazione	Accuracy	Precision	Recall	F1-score	Dimensione
Modello iniziale	84.29%	84.56%	84.29%	84.24%	23.41 MB
Statica	26.03%	32.49%	26.03%	25.33%	5.87 MB
Dinamica	84.25%	84.54%	84.25%	84.21%	9.24 MB

Tabella 4.3: Risultati prodotti con le quantizzazioni del framework PyTorch Mobile per il dataset CIFAR-10

La Tabella 4.4 riporta il consumo energetico richiesto dai modelli durante la fase di evaluation. Si nota che, in questo contesto, la Dynamic Quantization richiede più tempo per completare l'evaluation, e quindi ha un impatto ambientale maggiore. Pertanto, possiamo dedurre che la scelta tra le tecniche di quantizzazione per il dataset CIFAR-10 dipende principalmente dalla disponibilità di spazio di memoria sui microcontrollori utilizzati. Nel caso in cui lo spazio di memoria sia un fattore non limitante, la Dynamic Quantization potrebbe essere la scelta ideale. Tuttavia, se l'obiettivo primario è ottenere una maggiore efficienza energetica, allora è preferibile optare per la Static Quantization.

Quantizzazione	Durata	Emissioni di CO_2	Consumo energetico
Statica	0.91 min	0.2041 g	0.71497 Wh
Dinamica	8.35 min	1.8761 g	6.5713 Wh

Tabella 4.4: Risultati prodotti con la libreria CodeCarbon per l'evaluation dopo le quantizzazioni con PyTorch Mobile applicate al dataset CIFAR-10

I risultati sull'accuratezza dei modelli, dopo l'applicazione delle tecniche di quantizzazione fornite da TensorFlow Lite, sono presentati nella Tabella 4.5. In particolare, utilizzando la Dynamic Range Quantization e la Full Integer Quantization, si è ottenuta una riduzione delle dimensioni del modello del 91.64%, mentre con la Float16 Quantization dell'83.35%. Inoltre, è interessante notare che la Float16 Quantization non ha avuto un impatto negativo sull'accuratezza, mentre la Full Integer Quantization ha comportato una diminuzione del 31.9% dell'accuratezza del modello. Dato che la Dynamic Range Quantization consente di ottenere un'elevata accuratezza e una dimensione del modello ottimale, è sicuramente la tecnica di quantizzazione migliore in questo caso.

Quantizzazione	Accuracy	Precision	Recall	F1-score	Dimensione
Modello iniziale	80.89%	81.32%	80.89%	80.79%	70.29 MB
Dynamic Range	80.41%	80.82%	80.41%	80.39%	5.88 MB
Full Integer	48.58%	56.52%	48.58%	48.17%	5.89 MB
Float16	80.48%	80.84%	80.48%	80.45%	11.71 MB

Tabella 4.5: Risultati prodotti con le quantizzazioni del framework TensorFlow Lite per il dataset CIFAR-10

La Tabella 4.6 presenta il consumo energetico prodotto dai modelli nella fase di evaluation, evidenziando che la tecnica più efficiente dal punto di vista energetico risulta essere la Full Integer Quantization. Ma possiamo concludere che la scelta tra le diverse tecniche di quantizzazione dipende principalmente dalla disponibilità di spazio di memoria. Se lo spazio di memoria non è molto vincolante, la migliore opzione per il trade-off tra accuratezza e consumo energetico è la Float16 Quantization.

Quantizzazione	Durata	Emissioni di CO_2	Consumo energetico
Dynamic Range	2.15 min	0.4831 g	1.69197 Wh
Full Integer	1.32 min	0.2969 g	1.0397 Wh
Float16	1.43 min	0.3223 g	1.1289 Wh

Tabella 4.6: Risultati prodotti con la libreria CodeCarbon per l'evaluation dopo le quantizzazioni con TensorFlow Lite applicate al dataset CIFAR-10

Tra le diverse tecniche di quantizzazione esaminate per il dataset CIFAR-10, emerge che la migliore in termini di accuratezza è la Dynamic Quantization mentre la migliore dal punto di vista energetico è la Static Quantization, entrambe fornite da PyTorch Mobile. Tuttavia, il miglior equilibrio tra accuratezza e consumo energetico è ottenuto attraverso le tecniche di quantizzazione fornite da TensorFlow Lite.

4.3.2 Analisi dei risultati prodotti per il dataset SVHN

I risultati presentati nella Tabella 4.7 mostrano i risultati ottenuti applicando le tecniche di quantizzazione di PyTorch Mobile. In particolare, con la Static Quantization si ottiene una riduzione delle dimensioni del modello del 73.76%, mentre con la Dynamic Quantization la riduzione è del 35.75%. Inoltre, è importante notare che la Static Quantization ha un impatto significativo sull'accuratezza del modello del 72.2%. In questo contesto, sembra preferibile utilizzare la Dynamic Quantization, anche perché non richiede un notevole spazio di memoria e mantiene un livello accettabile di accuratezza del modello.

Quantizzazione	Accuracy	Precision	Recall	F1-score	Dimensione
Modello iniziale	87.03%	85.45%	84.08%	83.04%	2.21 MB
Statica	14.83%	58.59%	16.98%	14.02%	0.58 MB
Dinamica	87.03%	85.45%	84.08%	83.04%	1.43 MB

Tabella 4.7: Risultati prodotti con le quantizzazioni del framework PyTorch Mobile per il dataset SVHN

Invece, i risultati relativi alla misurazione energetica dei modelli in fase di evaluation sono riportati nella Tabella 4.8, la quale mostra il minor consumo energertico richiesto dalla Static Quantization. Tuttavia, è importante notare che la Static Quantization, nonostante il suo minor consumo energetico, comporta una significativa riduzione dell'accuratezza del modello. Pertanto, in questo caso, è preferibile utilizzare la Dynamic Quantization, che, pur richiedendo un consumo energetico leggermente maggiore, mantiene un buon livello di accuratezza del modello.

Quantizzazione	Durata	Emissioni di CO_2	Consumo energetico
Statica	1.04 min	0.3721 g	0.8220 Wh
Dinamica	4.28 min	1.526 g	3.3717 Wh

Tabella 4.8: Risultati prodotti con la libreria CodeCarbon per l'evaluation dopo le quantizzazioni con PyTorch Mobile applicate al dataset SVHN

I risultati presentati nella Tabella 4.9 riguardano i modelli creati attraverso le quantizzazioni disponibili in TensorFlow Lite. In particolare, la Dynamic Range Quantization ha mantenuto un alto livello di accuratezza, mentre la Float16 Quantization non ha mostrato variazioni significative in termini di accuratezza. D'altra parte, la Full Integer Quantization ha comportato una notevole riduzione dell'accuratezza dei modelli. In generale, i modelli hanno subito una riduzione delle dimensioni del 91.39%, ad eccezione della Float16 Quantization che ha ridotto la dimensione del 83.04%. Complessivamente, il modello migliore sembra essere quello quantizzato tramite la Dynamic Range Quantization, poiché ha una dimensione ridotta ma conserva un'alta accuratezza.

Quantizzazione	Accuracy	Precision	Recall	F1-score	Dimensione
Modello iniziale	95.97%	95.49%	95.68%	95.57%	6.72 MB
Dynamic Range	95.93%	95.47%	95.63%	95.53%	0.57 MB
Full Integer	6.7%	0.67%	10%	1.26%	0.57 MB
Float16	95.97%	95.49%	95.68%	95.57%	1.11 MB

Tabella 4.9: Risultati prodotti con le quantizzazioni del framework TensorFlow Lite per il dataset SVHN

La Tabella 4.10 riporta il consumo energetico dei modelli durante la fase di evaluation. In questo contesto, è evidente che la Float16 Quantization ha un minor impatto ambientale, ed è quindi il modello che rispetta maggiormente il trade-off tra accuratezza e consumo energetico.

Quantizzazione	Durata	Emissioni di CO_2	Consumo energetico
Dynamic Range	1.84 min	0.2014 g	1.4518 Wh
Full Integer	1.08 min	0.1178 g	0.8488 Wh
Float16	0.91 min	0.0997 g	0.7186 Wh

Tabella 4.10: Risultati prodotti con la libreria CodeCarbon per l'evaluation dopo le quantizzazioni con TensorFlow Lite applicate al dataset SVHN

Considerando tutti i risultati sopra riportati, è interessante notare che l'accuratezza peggiore si ottiene con la Static Quantization di PyTorch Mobile e con la Full Integer Quantization in TensorFlow Lite. Questo potrebbe essere dovuto alla forte quantizzazione dei modelli, poiché queste tecniche sono progettate per ottenere dimensioni molto ridotte dei modelli. Al contrario, la tecnica che conserva maggiormente l'accuratezza dei modelli in PyTorch Mobile è la Dynamic Quantization di PyTorch Mobile, mentre in TensorFlow Lite le tecniche migliori sono la Dynamic Range Quantization e la Float16 Quantization. Riguardo all'impatto ambientale, in PyTorch Mobile la Static Quantization sembra avere un minor impatto ambientale, mentre in TensorFlow Lite sia la Full Integer Quantization che la Float16 Quantization forniscono risultati soddisfacenti dal punto di vista energetico. Pertanto, la tecnica che sembra offrire il miglior trade-off tra accuratezza e consumo energetico è la Float16 Quantization di TensorFlow Lite.

In conclusione, per ottenere un modello di Deep Learning per la classificazione di immagini che abbia un impatto ambientale limitato, è consigliabile seguire il processo di allenamento con TensorFlow, applicare la Float16 Quantization al modello e, successivamente, renderlo disponibile sui microcontrollori. Questo approccio permette di mantenere un buon equilibrio tra accuratezza e consumo energetico, contribuendo così a ridurre l'impatto ambientale per i modelli di Machine Learning.

CAPITOLO 5

Conclusioni

Nel corso dello studio, è stato condotto un confronto tra i due framework più diffusi nel campo del Machine Learning, PyTorch e TensorFlow. Le prestazioni di entrambi i framework sono state analizzate nell'ambito del Classical Machine Learning e del Tiny Machine Learning, in riferimento alla classificazione di immagini utilizzando i dataset CIFAR-10 e SVHN. Al fine di ottenere i modelli "tiny", ai modelli sviluppati nel contesto del Classical Machine Learning sono state applicate tecniche di quantizzazione per ridurne le dimensioni. Inoltre, è stato effettuato un dettagliato confronto tra questi modelli, valutandone l'accuratezza e il consumo energetico, con l'obiettivo di individuare la soluzione ottimale per la progettazione di modelli green a basso impatto ambientale.

Per contribuire alla progettazione di modelli di Machine Learning in riferimento alla Green AI, ci sono molteplici opportunità per ulteriori sviluppi futuri. Ad esempio, si potrebbero esplorare altre tecniche di riduzione della dimensionalità, come il pruning, oppure sfruttare librerie specializzate per ridurre la dimensione dei modelli. Questo approccio consentirebbe di enfatizzare ulteriormente i vantaggi energetici e ambientali, contribuendo a ottimizzare i modelli in modo più adatto alle loro caratteristiche specifiche, anche attraverso la combinazione di diverse tecniche. Inoltre,

sarebbe opportuno misurare direttamente il consumo energetico sui microcontrollori (come Arduino o Raspberry Pi), al fine di ottenere dati più accurati e applicabili a scenari reali. In aggiunta, potrebbe essere utile valutare i modelli di Classical Machine Learning e Tiny Machine Learning anche mediante la comparazione del numero di FLOPs (operazioni in virgola mobile al secondo), per mettere in luce le differenze nella complessità computazionale dei modelli. Inoltre, per ampliare l'orizzonte di ricerca e contribuire alla promozione di modelli di Green Machine Learning in settori diversificati, si potrebbero esplorare applicazioni di Machine Learning su tematiche diverse dalla classificazione di immagini, oppure considerare la verifica di modelli pre-addestrati e compiti di Object Detection.

Bibliografia

[1] V. Rajapakse, I. Karunanayake, and N. Ahmed, "Intelligence at the extreme edge: A survey on reformable tinyml," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–30, 2023. (Citato alle pagine iii, 6, 7 e 8)

[2] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 517–532, 2021. (Citato alle pagine iv, 6, 7 e 9)

[3] S. Georgiou, M. Kechagia, T. Sharma, F. Sarro, and Y. Zou, "Green ai: Do deep learning frameworks have different costs?" in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1082–1094. (Citato alle pagine 4 e 22)

[4] R. Verdecchia, J. Sallou, and L. Cruz, "A systematic review of green ai," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, p. e1507, 2023. (Citato alle pagine 4 e 5)

[5] S. A. Budennyy, V. D. Lazarev, N. N. Zakharenko, A. N. Korovin, O. Plosskaya, D. V. Dimitrov, V. Akhripkin, I. Pavlov, I. V. Oseledets, I. S. Barsola *et al.*, "Eco2ai: carbon emissions tracking of machine learning models as the first step towards

sustainable ai," in *Doklady Mathematics*, vol. 106, no. Suppl 1. Springer, 2022, pp. S118–S128. (Citato a pagina 5)

[6] P. Warden and D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers*. O'Reilly, 2020. [Online]. Available: https://books.google.it/books?id=sB3mxQEACAAJ (Citato a pagina 5)

[7] R. Immonen, T. Hämäläinen *et al.*, "Tiny machine learning for resource-constrained microcontrollers," *Journal of Sensors*, vol. 2022, 2022. (Citato alle pagine 7 e 8)

[8] M. Sailesh, K. Selvakumar, and N. Prasanth, "A novel framework for deployment of cnn models using post-training quantization on microcontroller," *Microprocessors and Microsystems*, vol. 94, p. 104634, 2022. (Citato a pagina 9)

[9] S. Ghamari, K. Ozcan, T. Dinh, A. Melnikov, J. Carvajal, J. Ernst, and S. Chai, "Quantization-guided training for compact tinyml models," *arXiv preprint arXiv:2103.06231*, 2021. (Citato a pagina 9)

[10] J. D. De Leon and R. Atienza, "Depth pruning with auxiliary networks for tinyml," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 3963–3967. (Citato a pagina 10)

[11] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012. (Citato a pagina 14)

[12] D. P. Kingma and J. L. Ba, "Adam: Amethod for stochastic optimization." (Citato a pagina 17)

[13] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/f2925f97bc13ad2852a7a551802feea0-Paper.pdf (Citato a pagina 17)

Ringraziamenti

In conclusione, desidero esprimere il mio sincero ringraziamento a tutte le persone che hanno contribuito al raggiungimento di questo obiettivo.

Ringrazio il mio relatore Fabio Palomba, il quale, grazie alla sua competenza e passione, mi ha permesso di intraprendere questo percorso, consentendomi di arricchire le mie conoscenze ed aprirmi a nuove prospettive.

Ringrazio i miei correlatori Vincenzo De Martino e Stefano Lambiase per avermi costantemente guidato e supportato lungo questo percorso. Inoltre, vorrei esprimere la mia gratitudine per la vostra puntualità, la vostra disponibilità e i preziosi consigli che sono stati per me un incoraggiamento fondamentale.

Ringrazio i miei genitori per avermi permesso di inseguire i miei sogni, dedicandogli spazio anche nei momenti più bui degli ultimi anni. Anche se non vi ho sempre dato un motivo per credere in me, voi non avete mai smesso di incoraggiarmi e sostenermi. É solo grazie a voi se ho realizzato tutto questo e posso essere orgogliosa della persona che sono diventata.

Ringrazio Debora, la mia sorellona che è un po' anche il mio scudo. Mi hai insegnato a farmi superare le sfide e mantenere viva la mia fiducia in me stessa, indipendentemente dalle circostanze. Grazie per tutto quello che sei per me.

Ringrazio Biagio, ormai un fratello maggiore, per ogni momento trascorso insieme e per esserci in qualsiasi situazione.

Ringrazio Christian, per tutta la felicità e spensieratezza che mi trasmette, permettendomi sempre di ritornare bambina. La tua presenza, anche se da così poco, è il regalo più bello che potessi desiderare.

Ringrazio il mio splendido nonno, per tutti gli insegnamenti che mi hai donato e per avermi sempre incoraggiata nelle scelte che ho fatto. Sei la mia fonte di forza, e le tue preziose lezioni di vita saranno sempre una parte importante di me.

Ringrazio zia e zio, la mia seconda casa. Grazie per esserci in qualsiasi situazione e per tutte le pause studio che mi hanno permessa di distrarmi e motivarmi.

Ringrazio Mariapia e Giuseppina, per essere costantemente al mio fianco e per tutti i momenti trascorsi insieme, anche tramite videochiamate, che mi hanno permesso di rendere più leggere le mie giornate.

Ringrazio Marco, che mi sopportata e supportata in ogni istante, trovando sempre il modo di capire i miei bisogni. Mi hai accompagnata in questo percorso, condividendo con me i miei traguardi e rincuorandomi durante i momenti più difficili. Grazie per aver ascoltato i miei silenzi.

Ringrazio Ilenia, il mio punto di riferimento in ogni circostanza, che non ha mai smesso di credere in me, ma mi ha costantemente incoraggiata. Queste semplici parole non possono nemmeno avvicinarsi a grazie che ti devo davvero.

Ringrazio Natalia per essere stata al mio fianco lungo tutto il percorso, dall'infanzia all'università, affrontando insieme le sfide che la vita ci ha presentato, e per aver pazientemente ascoltato i miei sfoghi quotidiani.

Ringrazio Gerarda, per avermi sostenuto con costanza e per essere sempre al mio fianco, festeggiando con me ogni traguardo raggiunto.

Ringrazio Dina, per avermi trasmesso energia positiva ogni volta, trovando sempre il momento adatto per la pausa al bar.

Ringrazio Maria Pia, Raffaella, Federico, Lino, Angelo, Rocco e Raffaele per tutti i momenti trascorsi insieme.

Ringrazio Simona, per aver condiviso con me quasi tutto di questo percorso universitario, dalle ansie pre-esame agli infiniti casini combinati insieme. Da sempre hai avuto la capacità di comprendermi con un solo sguardo, e il tuo sostegno nel tempo è diventato sempre più fondamentale.

Ringrazio Francesco, per tutte le discussioni costruttive che ci hanno premesso di crescere e maturare. Ma soprattutto grazie per aver condiviso con me anche tutti i momenti post-esame passati al bar.

Ringrazio Luigina, per tutti i momenti passati insieme tra una lezione e l'altra, e per aver ascoltato per ore le mie difficoltà.

Ringrazio Roberto, per tutte le ore di studio trascorse insieme, che tra una risata e l'altra hanno permesso di rendere leggera ogni difficoltà.

Infine, ringrazio Tetta, per tutta l'energia, la forza e l'amore che mi ha donato.

Questa tesi ha contribuito a piantare un albero in Ghana tramite il progetto Treedom.

https://www.treedom.net/it/user/sesalab/event/sesa-random-forest