Università degli Studi di Milano

Dipartimento di Matematica "Federigo Enriques"

Reti neurali quantizzate per il riconoscimento di keywords

Lorenzo Gentile
Matricola: 966196

Relatore: Prof. Giovanni Naldi

Anno Accademico 2022/2023

Indice

1 Introduzione

Il riconoscimento vocale automatico (KeyWord Spotting, KWS) è un campo di ricerca che si occupa di identificare parole chiave all'interno di un flusso audio. Questo processo è molto utile per applicazioni come gli assistenti vocali, che devono essere in grado di riconoscere comandi come "Hey Siri" o "Ok Google" che attivino il loro funzionamento.

Per quanto riguarda il lato software, attualmente l'approccio più comune si basa sull'utilizzo di reti neurali, poiché portano a risultati molto migliori rispetto ai metodi tradizionali di analisi del segnale [1]. I modelli più promettenti in termini di accuratezza delle predizioni sono quelli basati architetture convoluzionali. Questo però ha anche degli svantaggi, primo fra tutti l'alto consumo energetico di questi modelli: siccome gli assistenti vocali devono essere continuamente in ascolto e spesso sono implementati su hardware con memoria e potenza di calcolo molto limitati, l'efficienza diventa un fattore cruciale.

Per quanto riguarda l'hardware, un'alternativa promettente ai tradizionali microcontrollori è rappresentata da nuove architetture in grado di eseguire operazioni in memoria (Compute-in-memory, CIM). Queste si prestano molto bene ad applicazioni come l'inferenza di reti neurali (già addestrate), e hanno un consumo energetico molto inferiore rispetto alle architetture tradizionali.

L'obiettivo di questo progetto è quello di sviluppare una rete neurale per il Keyword Spotting che miri ad avere un consumo energetico e di memoria molto limitato e che possa essere implementata su un'architettura CIM.

2 Contesto

2.1 Moltiplicazione matrice-vettore in memoria

La scheda utilizzata nel progetto monta dei banchi di memoria di tipo RRAM (Resistive Random Access Memory). Queste memorie (ancora in fase di sviluppo) introducono un diverso paradigma di calcolo rispetto alle tradizionali architetture di Von Neumann. Anziché utilizzare la memoria solamente per immagazzinare i dati e delegare alla CPU il compito di eseguire operazioni su questi, l'idea è eseguire specifiche operazioni direttamente all'interno della memoria. Questo riduce drasticamente l'overhead dovuto al trasferimento dei dati dalla memoria al processore e viceversa. L'operazione più comune che può essere svolta in memoria è la moltiplicazione matrice-vettore (MVM), che sta alla base di numerosi algoritmi numerici molto utilizzati nelle applicazioni.

Il principio di funzionamento è il seguente: le celle di memoria, che possono essere modellizzate come resistenze, sono disposte in una griglia rettangolare (Crosspoint Array) [Fig. 1]. Questo definisce una matrice di resistenze R_{ij}, o meglio di conduttanze $G_{ij} = 1/R_{ij}$, che indichiamo con G. Le conduttanze vengono settate con i valori numerici della matrice che si vuole moltiplicare e rappresentano quindi il dato immagazzinato nella cella di memoria. Un vettore di voltaggi $V = [V_1, V_2, ..., V_n]$ viene applicato alle colonne della griglia e come conseguenza delle leggi di Ohm e Kirchhoff si ottiene come output un vettore di correnti $I = [I_1, I_2, ..., I_m]$, che coincide con il prodotto matrice-vettore $G \cdot V$. [2]

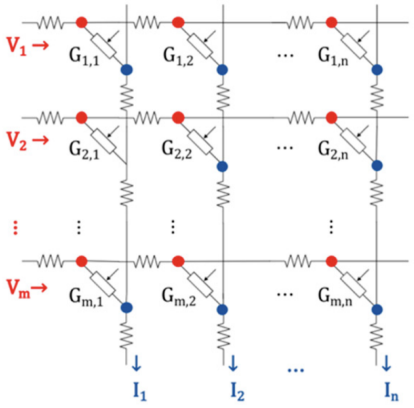

Figura 1: Disposizione delle celle per moltiplicazione matrice-vettore

2.2 Vantaggi e svantaggi

La moltiplicazione avviene nel dominio analogico, e questo porta molti vantaggi dal punto di vista dell'efficienza temporale : teoricamente il tempo di calcolo non dipende dalle dimensioni dell'input, contrariamente a quanto succede con un algoritmo tradizionale per MVM, la cui complessità temporale dipende è quadratica nelle dimensioni dell'input.

Questo approccio presenta anche degli svantaggi: le operazioni nel dominio analogico sono svolte con un grado di precisione limitato, il che è dovuto a diversi fattori, tra cui la variabilità dei valori delle resistenze nel tempo (drift), le conversioni tra analogico e digitale e vari effetti di parassitismo.
Inoltre le singole celle non possono assumere valori di conduttanza arbitrari, ma solo un numero finito di valori discreti. Questo comporta la necessità di un processo di quantizzazione dell'input.

2.3 Inferenza di reti neurali su architetture CIM

Una delle applicazioni principali delle architetture CIM è l'inferenza di reti neurali. Partendo da un modello già addestrato, i pesi della rete vengono scritti nelle celle di memoria sotto forma di conduttanze e le moltiplicazioni matrice-vettore vengono eseguite in memoria. I pesi della rete non devono cambiare durante l'inferenza, quindi in teoria dovrebbero semplicemente essere scritti all'inizio del processo. Nella pratica questi devono essere aggiornati con una certa frequenza per questioni di rumore.
Per quanto detto prima si presenta quindi la necessità di quantizzare non solo i pesi della rete, ma anche gli input di ogni layer.

Le operazioni di convoluzione devono essere mappate in moltiplicazioni matrice-vettore, mentre tutte le altre operazioni che avvengono nella rete (attivazioni, normalizzazioni, pooling, etc...), che non sono riducibili a operazioni lineari, devono essere svolte nel microprocessore in maniera digitale. Questo comporta che i dati debbano essere continuamente trasformati dal dominio analogico a quello digitale e viceversa. Questo continuo passaggio è necessario in tutti i casi, infatti senza di questo gli errori di calcolo vengono accumulati e l'output del modello risulterebbe sbagliato.
Nonostante ciò la maggior parte del workload durante l'inferenza di una rete corrisponde alle operazioni svolte nei layer densi e convoluzionali; quindi questo approccio è comunque più efficiente dal punto di vista computazionale rispetto agli approci completamente digitali.

2.4 Specifiche della scheda

La scheda utilizzata nel progetto è stata sviluppata dal gruppo di ricerca del Prof. Ielmini presso il Dipartimento di Elettronica, Informazione e Bioingegneria del Politecnico di Milano.
La scheda monta quattro blocchi di RRAM, ciascuna con 20000 celle di memoria, per un totale di 80000 celle. E' poi presente un microprocessore che gestisce le memorie e svolge calcoli ausiliari. L'insieme di valori di conduttanze (misurata in microSiemens) che le celle possono assumere viene continuamente modificato dai ricercatori del laboratorio, i quali cercano di trovare il giusto equilibrio tra espressività e precisione. Più sono i valori possibili, più aumenta il rischio che gli errori non permettano la distinzione tra stati diversi delle celle. Una possibile configurazione conta 11 livelli di conduttanza, ugualmente spaziati nell'intervallo $[-50\mu S, 50\mu S]$.

Il vettore dei voltaggi V, che deve essere moltiplicato con la matrice delle conduttanze G, deve essere un vettore binario. Questo perchè in questa particolare architettura, ogni riga della griglia di celle funziona come un interruttore. Quindi ad ogni colpo di clock, un singolo crosspoint array riesce ad eseguire una moltiplicazione tra un vettore binario e una matrice di valori quantizzati. Per incrementare la precisione in bit dei vettori di input, è possibile eseguire più moltiplicazioni in sequenza e sommare i risultati. Per un input a n bit l'operazione richiede n colpi di clock :

$$V = V_1 + 2 \cdot V_2 + ... + 2^{n-1} \cdot V_{n-1} \implies G \cdot V = G \cdot V_1 + 2 \cdot G \cdot V_2 + ... + 2^{n-1} \cdot G \cdot V_n$$

dove i V_i sono vettori binari.

3 Metodologie

3.1 Dataset

Il modello è stato scritto in Python utilizzando Tensorflow, una delle più diffuse librerie open source per il deep learning. Il dataset utilizzato per il progetto è il Google Speech Commands Dataset v2, che contiene 105829 file audio (formato .wav) di 35 parole diverse, pronunciate da diverse persone. Ogni sample ha una durata di 1 secondo e una frequenza di campionamento di 16000 Hz.
Date le dimensioni ridotte del modello, che deve avere un massimo di 80000 parametri a bassa precisione, sono state selezionate solo 5 categorie di parole: "yes", "no", "up", "down" e "left". Questo ha ridotto il numero totale di esempi a circa 4000, che sono state suddivise come segue: 80% training set, 10% validation set e 10% test set.

3.2 Data Augmentation

Sono state applicate al dataset diverse trasformazioni [Fig. 2]. Questo è stato fatto per due motivi: da un lato per aumentare la varietà dei dati, dall'altro per simulare meglio le condizioni reali in cui il modello deve operare. Infatti, quando si utilizza un assistente vocale, la parola pronunciata dall'utente può essere distorta o shiftata nel tempo. Queste trasformazioni contribuiscono anche a far generalizzare meglio il modello e a renderlo più robusto.
In particolare :

1. **Aggiunta di rumore**. Sono stati aggiunti due tipi di rumore: rumore bianco e rumore rosa. Il rumore bianco contiene tutte le frequenze dello spettro udibile in egual misura, mentre il rumore rosa ha una distribuzione di potenza che decresce linearmente con la frequenza.

2. **Shift temporale**. Il segnalo audio è stato shiftato in avanti o indietro nel tempo tramite dei piccoli incrementi casuali. Questo aiuta il modello ad essere tempo-invariante e riconoscere le parole anche tagliate o pronunciate a metà.

3. **Pitch shift**. Anche il pitch è stato modificato in maniera casuale, per far riconoscere meglio al modello voci e tonalità diverse.

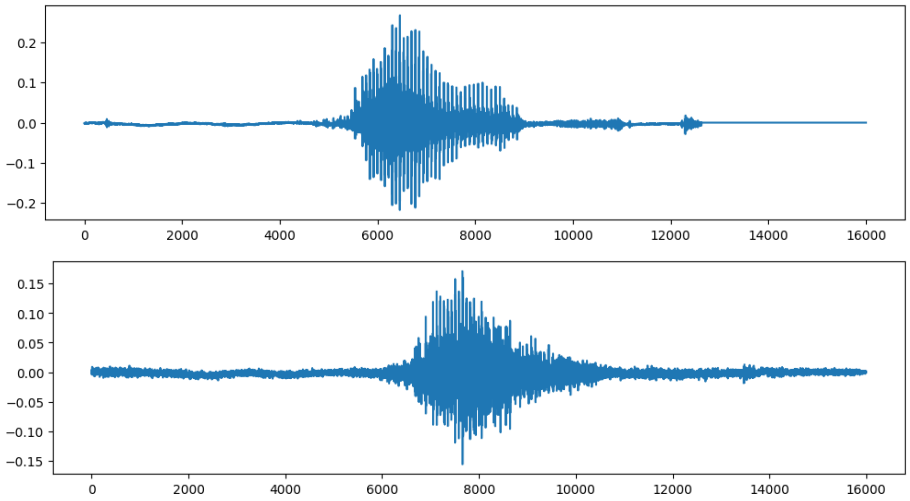

Figura 2: Stesso segnale audio ("left") prima e dopo le trasformazioni

3.3 Trasformate

Per il Keyword Spotting su dispositivi con risorse limitate, l'architettura che ha dato i migliori risultati è quella convoluzionale [3]. Con questi modelli si raggiunge una precisione alta con un numero molto limitato di parametri rispetto ai modelli completamente connessi.
E' necessario trasformare il segnale audio in una sua rappresentazione bidimensionale affinchè possa essere utilizzato come input di una rete convoluzionale.
In questo progetto sono state provate 3 trasformate diverse, così da poterle confrontare e scegliere quella che dà i risultati migliori.

3.3.1 STFT

La trasformata di Fourier (discreta) prende in input un segnale e restituisce la sua rappresentazione (monodimensionale) nel dominio delle frequenze. Essa non contiene più nessuna informazione sul tempo (esprime quali frequenze sono presenti, ma non quando sono presenti nel segnale), quindi nella sua forma "pura" non è molto adatta al riconoscimento di parole. D'altro canto, la trasformata di Fourier a tempo breve (Short Time Fourier Transform, STFT) cerca di mantenere anche informazioni sul tempo: il segnale viene suddiviso in finestre temporali e per ognuna di queste viene calcolata la trasformata di Fourier. Si ottiene una matrice di frequenze complesse e per ogni elemento di questa matrice si calcola il modulo. La rappresentazione bidimensionale che ne risulta è detta spettrogramma del segnale originale [Fig. 3].

In questo caso la dimensione degli spettrogrammi ottenuti è 124×129.

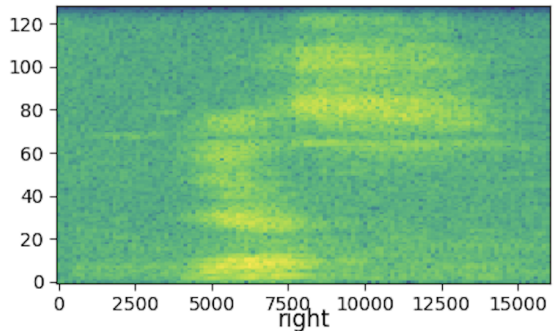

Figura 3: Spettrogramma di un esempio del dataset

3.3.2 MFCC

Un'altro approccio molto utilizzato per il riconoscimento vocale sono i coefficienti spettrali Mel (Mel Frequency Cepstral Coefficients, MFCC). Questi sono calcolati a partire dallo spettrogramma del segnale audio, quindi sono più costosi da computare rispetto alla STFT. Dopo aver applicato un banco di filtri allo spettrogramma e averlo quindi suddiviso in bande di frequenza, si calcola il logaritmo della potenza di ogni banda. A questo punto viene calcolata la trasformata di Fourier inversa di queste potenze, e i coefficienti che ne risultano sono i coefficienti cercati.
Questo metodo tiene conto della percezione umana delle frequenze, che non è lineare ma segue una scala logaritmica. Inoltre, i coefficienti MFCC sono meno sensibili alle variazioni di pitch e di volume rispetto allo spettrogramma.
Questa rappresentazione è più compatta rispetto agli spettrogrammi : la dimensione di una singola immagine è 59×13. [Fig. 4]

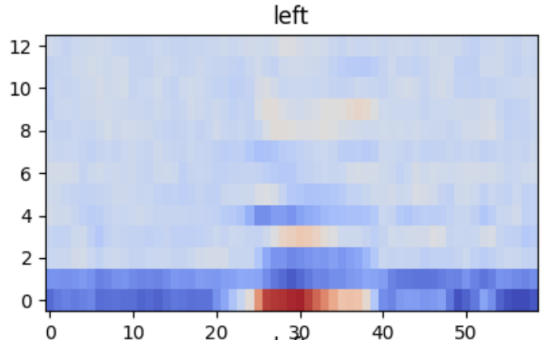

Figura 4: Coefficienti spettrali Mel di un esempio del dataset

3.3.3 Trasformata Wavelet

Il motivo per cui la trasformata di Fourier non conserva nessuna informazione sul tempo è che decompone il segnale utilizzando funzioni analizzanti che non sono localizzate nel tempo.
Per esempio le serie di Fourier tradizionali decompongono un segnale $f(t) \in L^2([-T,T])$ sulla base ortonormale $\{e^{i\frac{2\pi}{T}nt} = cos(\frac{2\pi}{T}nt) + i \cdot sin(\frac{2\pi}{T}nt)\}_{n\in\mathbb{Z}}$. Ciascuna di queste funzioni g è periodica e oscilla lungo tutto l'asse dei tempi, quindi il prodotto scalare $\langle f, g\rangle$ indica solo quanto il segnale f sia correlato con la funzione g, ma non indica quando questa correlazione avviene.
L'idea della trasformata wavelet è quella di utilizzare per la decomposizione una base di funzioni localizzate nel tempo. Per costruire una base di $L^2(\mathbb{R})$ con questa proprietà, si parte da una funzione $\psi : \mathbb{R} \to \mathbb{C}$ detta wavelet madre [Fig. 5], che deve soddisfare due proprietà principali:

1. $\int_{-\infty}^{+\infty} \psi(t)dt = 0$, ovvero la wavelet madre deve avere media nulla. Questa condizione impone a ψ di essere una funzione "oscillante": intuitivamente le aree positive e negative sottese dal grafico devono essere uguali.

2. $\psi \in L^2(\mathbb{R})$, ovvero $\int_{-\infty}^{+\infty} |\psi(t)|^2 dt < \infty$. Si dice che ψ ha energia finita: questa condizione è quella che garantisce che la funzione sia localizzata nel tempo. Per esempio le funzioni trigonometriche come $sin(t)$ non hanno energia finita.

A questo punto si definiscono le traslazioni e dilatazioni di ψ:

$$\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k) \tag{1}$$

dove $j, k \in \mathbb{Z}$.
Si dimostra che $\{\psi_{j,k}\}_{j,k\in\mathbb{Z}}$ è una base ortonormale di $L^2(\mathbb{R})$. A differenza delle basi trigonometriche standard, gli elementi di questa base sono indicizzati da due parametri: variando j si ottengono "frequenze" diverse (in realtà si parla

in questo caso di "scale"), mentre variando k si ottengono le traslate di ψ. I coefficienti della decomposizione di f rispetto a valori diversi di k permettono quindi di identificare quando certe frequenze sono presenti nel segnale.

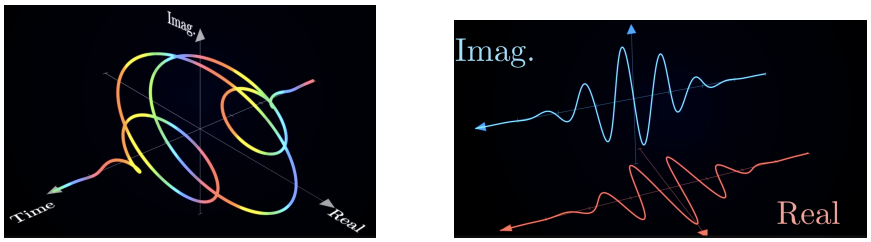

Figura 5: Morlet Wavelet - $\Psi(t) = e^{i\omega_0 t} e^{-t^2/2}$. A sinistra il grafico in $\mathbb{R} \times \mathbb{C}$, a destra le due componenti reale e immaginaria.

La rappresentazione di un segnale f rispetto ad un sistema di wavelet è bidimensionale: una dimensione rappresenta le scale, l'altra il tempo. Questa rappresentazione è detta scalogramma.
Esistono molte famiglie di wavelet, che differiscono per la forma della wavelet madre e sono adatte ad analizzare segnali con diverse caratteristiche (audio, scosse sismiche, ECG, etc...).
Per il progetto è stata utilizzata la wavelet di Haar, che è la piu semplice e ha il pregio di essere molto veloce da calcolare. Essa è definita come segue:

$$\psi(t) = \begin{cases} 1 & \text{se} \quad 0 \leq t < 1/2 \\ -1 & \text{se} \quad 1/2 \leq t < 1 \\ 0 & \text{altrimenti} \end{cases} \tag{2}$$

Gli scalogrammi ottenuti hanno dimensione 280×4. [Fig. 6]

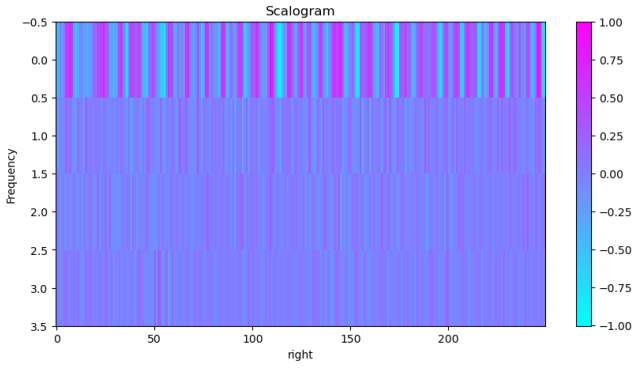

Figura 6: Scalogramma di un esempio del dataset

3.4 Quantizzazione

La quantizzazione di una rete neurale consiste nel ridurre la precisione dei parametri e delle attivazioni della rete, che normalmente occupano 32 bit. E' una tecnica che ha permesso negli ultimi anni di implementare modelli su hardware con risorse molto limitate [4]. Oltre a ridurre lo spazio necessario a memorizzare i parametri, l'aritmetica a precisione ridotta è molto più efficiente dal punto di vista computazionale.
Esistono due approcci principali per quantizzare una rete neurale:

1. **Post Training Quantization (PQT)**. La rete viene addestrata con precisione a 32 bit e successivamente i pesi vengono quantizzati con precisone ridotta. Questo metodo è molto semplice da implementare, ma porta ad una perdita di accuratezza non trascurabile, soprattutto se si utilizzano pochi bit per la quantizzazione.

2. **Quantization Aware Training (QAT)**. La rete viene addestrata con la precisione scelta. Questo metodo è più complesso da implementare, poiché bisogna modificare l'algoritmo di backpropagation per tener conto della quantizzazione, ma porta a risultati in genere migliori rispetto al primo metodo.

Per la quantizzazione è stato utilizzato larq [5], un modulo open source che estende Tensorflow aggiungendo dei layer specifici per la quantizzazione. Larq funziona nel seguente modo: ogni peso è memorizzato sia a 32 bit che quantizzato. Durante il training, il forward pass viene eseguito con i pesi quantizzati, mentre durante il backward pass vengono aggiornati i pesi a 32 bit (questo è necessario per calcolare i gradienti). A questo punto il valore quantizzato dei pesi viene ricalcolato a partire dal valore aggiornato a 32 bit.
Larq mette a disposizione diverse funzioni di quantizzazione (quantizzatori). Per le attivazioni è stato utilizzato il quantizzatore DoReFa a 3 bit [Fig. 7], che è stato sviluppato appositamente per reti neurali convoluzionali [6].
Tutti i quantizzatori disponibili in larq operano con un certo numero di bit n, quindi i valori quantizzati variano in 2^n livelli distinti. La scheda utilizzata nel progetto però necessita di un numero di livelli diverso in generale da una potenza di 2. Per risolvere questo problema, è stato utilizzato un quantizzatore custom per i pesi. Questo quantizzatore può essere configurato con un vettore arbitrario di livelli $L = [l_1, l_2, ..., l_k]$, dove k è il numero di livelli, prende in input un valore x e restituisce il valore l_i più vicino a x.
Per il calcolo del gradiente è stato utilizzato il metodo STE (Straight Through Estimator), che consiste nel propagare il gradiente senza modificarlo [7]. Questo è necessario perchè il gradiente reale di qualsiasi funzione di quantizzazione è nullo quasi ovunque, e quindi non è possibile utilizzare questo durante il back-

ward pass.

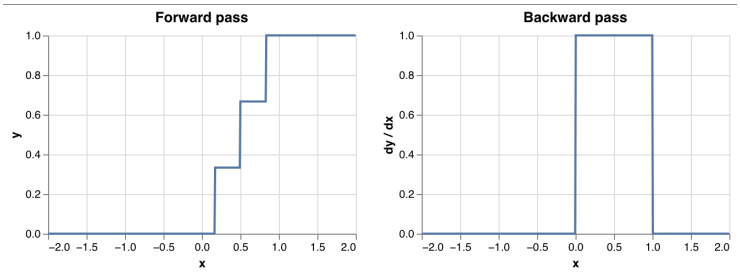

Figura 7: $q(x) = \begin{cases} 0 & x < \frac{1}{2n} \\ \frac{i}{n} & \frac{2i-1}{2n} < x < \frac{2i+1}{2n} \text{ for } i \in \{1, n-1\} \\ 1 & \frac{2n-1}{2n} < x \end{cases}$

3.5 Architettura del modello

Come già detto, il modello utilizzato è una rete neurale convoluzionale. Sono state provate svariate configurazioni, ma la migliore è risultata essere la seguente:

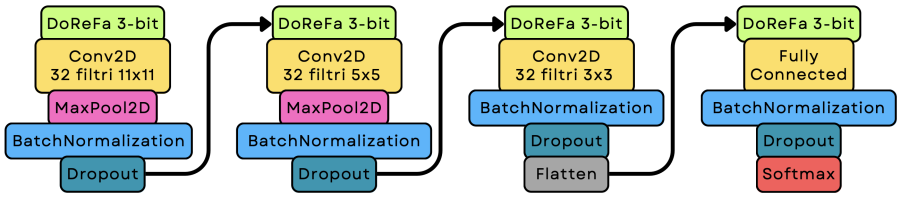

Figura 8: Architettura del modello

La rete è formata da 3 layer convoluzionali, ciascuno con 32 filtri di dimensione decrescente (11x11, 5x5, 3x3), seguiti da MaxPooling e un layer denso finale, il cui output è un vettore di 5 elementi (uno per ogni classe).
Prima di ogni layer denso o convoluzionale l'input viene quantizzato con il quantizzatore DoReFa a 3 bit. Sono stati poi aggiunti dei layer di Batch Normalization e dropout per ridurre l'overfitting e rendere il training più stabile.
Il modello conta 74,890 parametri trainabili (layer convoluzionali e denso), tutti quantizzati su 11 livelli uniformemente spaziati nell'intervallo [-1,1]. I restanti 202 parametri sono i pesi dei layer di Batch Normalization, che sono memorizzati a 32 bit, poiché le operazioni di questi layer devono avvenire nel digitale.

4 Risultati

4.1 Accuratezza del modello

Tra i 3 metodi esplorati per la trasformazione del segnale audio, quello che ha dato i risultati migliori è stato lo spettrogramma (STFT), con il quale è stata raggiunta una accuratezza del 95% sul test set.
Con gli altri due metodi il modello fa molta più fatica a convergere e l'accuratezza finale è molto più bassa (50-60%).
Questo è probabilmente dovuto all'estrema quantizzazione delle attivazioni: le immagini ottenute con la trasformata wavelet e i coefficienti MFCC sono molto più compatte rispetto agli spettrogrammi, quindi la quantizzazione a 3 bit ha un effetto più drastico e ne riduce di molto l'espressività.
Il numero di epoche necessario a raggiungere un'accuratezza soddisfacente è molto alto (circa 200), ma questo è tipico per le reti quantizzate, soprattutto per quelle a bassa precisione. La quantizzazione rende il processo di training molto più lento del normale perchè i pesi sono costretti ad assumere un insieme limitato di valori. Nonostante ciò, dopo abbasanza epoche, il modello riesce a convergere e raggiungere un'accuratezza molto alta.

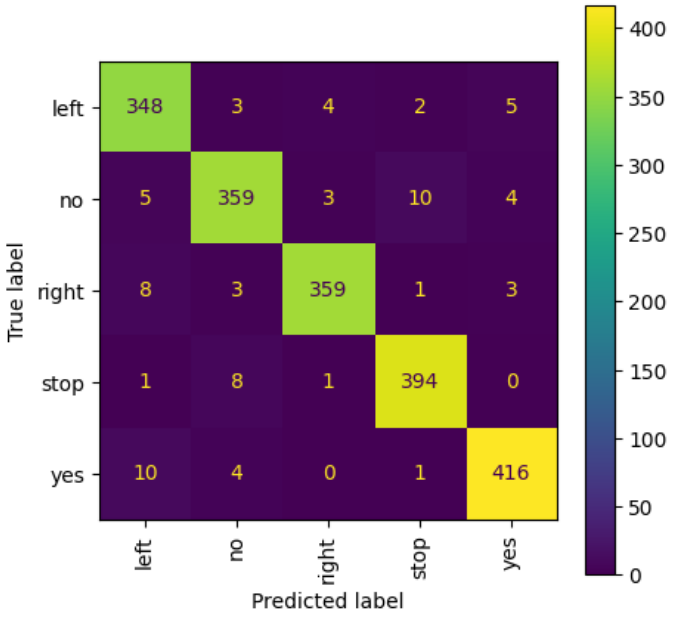

Figura 9: Matrice di confusione del modello sul test set

4.2 Output dei layer intermedi

Per capire meglio come funziona il modello, è stato analizzato l'output dei layer intermedi. Questo è stato utile anche per capire che impatto hanno le quantizzazioni sui dati e cercare di ridurre la perdita di informazione dovuta a questo. Il flusso dei dati è rappresentato nelle immagini seguenti.

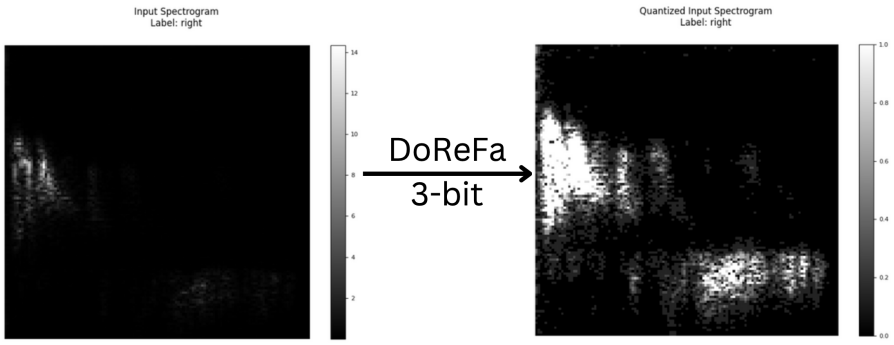

Figura 10: lo spettrogramma (124×129) viene quantizzato a 3 bit e passato al primo layer convoluzionale.

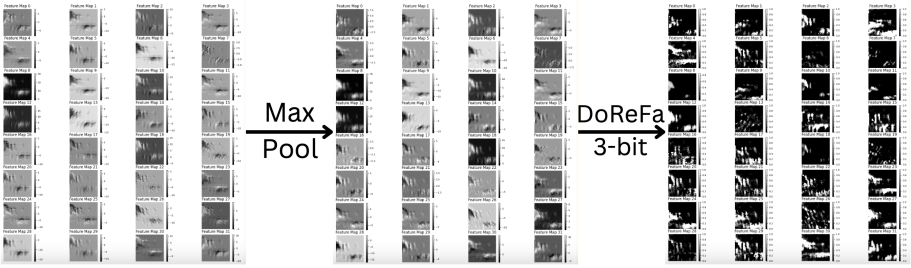

Figura 11: l'output del primo layer convoluzionale è un tensore di dimensioni $114 \times 119 \times 32$. A questo viene applicato un MaxPooling con finestra 3×3, che ne riduce le dimensioni a $38 \times 39 \times 32$, e viene successivamente riquantizzato prima di essere passato al layer convoluzionale succesivo.

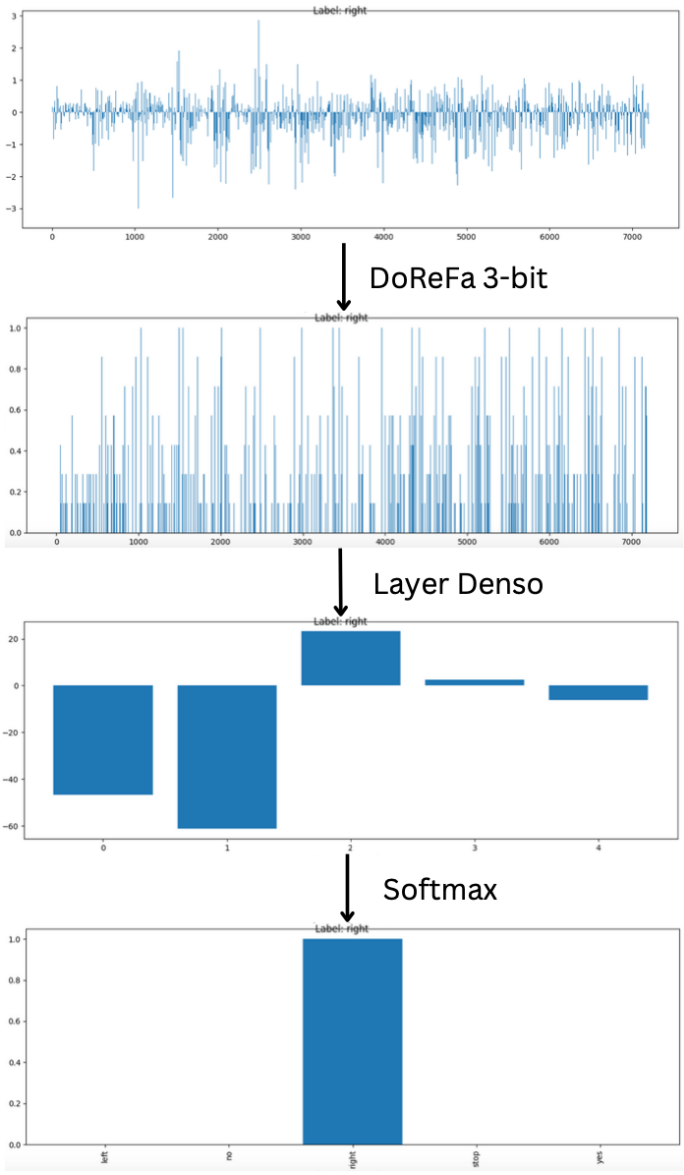

Figura 12: dopo il terzo layer convoluzionale, l'output (che a questo punto ha dimensioni $15 \times 15 \times 32$) viene compattato in un vettore monodimensionale con 7200 entrate e quantizzato prima di essere passato al layer denso. L'output di quest'ultimo è un vettore di 5 elementi, che dopo l'attivazione softmax diventa un vettore di 5 probabilità. Questo coincide con l'output del modello, che in questo esempio il modello ha correttamente classificato la parola "left".

5 Discussione e conclusioni

I risultati ottenuti sono promettenti e dimostrano che è possibile ottenere modelli per il KWS altamente quantizzati, senza incidere significativamente sull'accuracy. Il modello citato nel progetto (80k parametri) mantiene un'accuracy sopra il 90% anche se addestrato con 12 classi distinte di keywords. Aumentando il numero di parametri a circa 150k invece (con le stesse quantizzazioni) si riesce a raggiungere un'accuracy sopra il 90% con tutte le 34 classi del dataset.
Si possono stimare le dimensioni del modello in memoria. Per esempio 80k parametri quantizzati a 4 bit (16 livelli) occupano 40kB.

Inoltre il modello è un ottimo punto di partenza per un' implementazione sulla scheda sviluppata dal team del prof. Ielmini. In questo senso ci sono altri ostacoli da superare. Un problema è quello di mappare in maniera efficiente le operazioni della rete, come le convoluzioni, sulla memoria. La scheda è più veloce se le zone della memoria che vengono accedute in sequenza sono vicine tra loro, quindi è importante trovare un buon modo di organizzare i dati durante le operazioni.
Bisogna anche tenere conto dei parassitismi elettrici, in particolare di questi due tipi di rumore:

1. Ogni volta che un peso viene scritto nella memoria, il valore effettivo assunto dalla resistenza non coincide esattamente con il valore del peso, ma statisticamente segue una distribuzione normale centrata intorno ad esso. I pesi vengono quindi già scritti con un certo errore.

2. Ogni volta che una cella viene "attivata", cioè viene coinvolta in una moltiplicazione, il valore della resistenza cambia seguendo ancora una distribuzione normale. Questo effetto è molto più leggero del primo, ma avviene molto più spesso.

E' necessario testare se la rete sia robusta rispetto a questi rumori e in caso contrario mitigare il problema, per esempio tenendo conto dei parassitimi durante il training.

Bibliografia

[1] Yundong Zhang et al. "Hello Edge: Keyword Spotting on Microcontrollers". In: *CoRR* abs/1711.07128 (2017). arXiv: 1711.07128. URL: http://arxiv.org/abs/1711.07128.

[2] Zhong Sun and Ru Huang. “Time Complexity of In-Memory Matrix-Vector Multiplication”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 68.8 (2021). DOI: 10.1109/TCSII.2021.3068764.

[3] Carolina Parada Tara N. Sainath. “Convolutional Neural Networks for Small-footprint Keyword Spotting”. In: 2015, pp. 106–111. URL: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43969.pdf.

[4] Sek Chai Prateeth Nayak David Zhang. “Bit Efficient Quantization for Deep Neural Networks”. In: 2019. URL: https://arxiv.org/pdf/1910.04877.pdf.

[5] Lukas Geiger and Plumerai Team. “Larq: An Open-Source Library for Training Binarized Neural Networks”. In: *Journal of Open Source Software* 5 (Jan. 2020), p. 1746. DOI: 10.21105/joss.01746.

[6] Shuchang Zhou et al. *DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients*. 2018. arXiv: 1606.06160 [cs.NE].

[7] Penghang Yin et al. *Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets*. 2019. arXiv: 1903.05662 [cs.LG].