

Politecnico di Torino

Politecnico di Torino

Laurea Magistrale in Ingegneria Informatica - Automation and Intelligent Cyber-Physical Systems

A.a. 2025/2026

Sessione di laurea Luglio 2026

SentoMeglio

Deep learning per lo speech enhancement in tempo reale su smartphone come ausilio per persone con ipoacusia

Relatori:

Eros Pasero

Vincenzo Randazzo

Candidato:

Matteo Giuliani

Sommario

Per chi soffre di ipoacusia la difficoltà maggiore non è il silenzio, ma il rumore: in un ambiente affollato la voce dell'interlocutore si perde nel brusio di fondo. Gli apparecchi acustici, che si fondano su tecniche di elaborazione del segnale tradizionali e tarate su un rumore stazionario, danno benefici limitati in queste condizioni e restano dispositivi costosi. Questa tesi sviluppa un sistema di *speech enhancement* basato su reti neurali profonde che opera in tempo reale su un comune smartphone Android.

Il punto di partenza è il modello STFT-TCNN, una rete temporale convoluzionale che stima una maschera spettrale nel dominio tempo-frequenza. Su questo baseline il lavoro introduce diverse modifiche. Una funzione di costo percettiva, di ispirazione psicoacustica, dà più peso agli errori della rete nelle bande di frequenza a cui l'orecchio umano è più sensibile. Un modulo di compressione frequenziale a banda critica (ERB) riduce poi di circa quattro volte i parametri, preservando l'informazione percettivamente rilevante e abbassando di conseguenza anche il tempo di inferenza. Una riformulazione causale a stati espliciti, infine, elimina la dipendenza dai campioni futuri e abilita l'elaborazione frame-by-frame a latenza minima.

Il modello che ne risulta è reso fruibile all'utente attraverso SentoMeglio, l'applicazione Android sviluppata in questa tesi per eseguirne l'inferenza direttamente sul telefono.

Indice

Elenco delle tabelle VI

Elenco delle figure VIII

1 Introduzione 1
- 1.1 Ipoacusia . . . 1
- 1.2 Apparecchi acustici . . . 1
- 1.3 Cambio di paradigma . . . 2
- 1.4 Struttura della tesi . . . 3

2 Reti neurali 5
- 2.1 Tipologie di apprendimento . . . 5
- 2.2 Le Reti Neurali Artificiali . . . 7
  - 2.2.1 Funzioni di attivazione . . . 7
  - 2.2.2 Il ciclo di addestramento . . . 9
  - 2.2.3 Architetture principali di reti neurali . . . 10

3 Speech Enhancement 12
- 3.1 Masking e mapping . . . 13
- 3.2 Efficienza computazionale e vincoli implementativi . . . 14
- 3.3 Metriche e valutazione dei modelli . . . 15

4 Analisi nel dominio della frequenza 17
- 4.1 Proprietà in frequenza della voce . . . 17
- 4.2 La Short-Time Fourier Transform (STFT) . . . 18
- 4.3 Lo spettrogramma . . . 19
- 4.4 Parametri della STFT . . . 19

5 STFT-TCNN 21
- 5.1 Pipeline di elaborazione . . . 21
- 5.2 Struttura della rete . . . 23

5.2.1 Il blocco residuo . . . 23
5.2.2 La rete completa . . . 25
5.3 Configurazione di riferimento . . . 28
5.3.1 Il campo ricettivo . . . 28
5.3.2 Tabella di riferimento . . . 29

6 Loud-loss 30
6.1 Il modello psicoacustico e la scala Phon . . . 30
6.2 Formulazione matematica . . . 31
6.3 Variante con SI-SNR . . . 33
6.4 Risultati sperimentali . . . 34
6.4.1 Setup . . . 34
6.4.2 Raccolta dei dati . . . 34
6.4.3 Valutazione dei risultati . . . 35

7 Modulo ERB 37
7.1 Equivalent Rectangular Bandwidth . . . 37
7.2 Compressione delle feature e riduzione della complessità . . . 39
7.3 Formulazione matematica . . . 40
7.4 Collocazione nell'architettura della rete . . . 42
7.5 Risultati sperimentali . . . 42
7.5.1 Qualità della ricostruzione . . . 43
7.5.2 Costo su dispositivo . . . 43

8 Streaming real-time 45
8.1 Causalità e funzionamento real-time . . . 45
8.2 La perdita di causalità nella baseline . . . 46
8.3 Riformulazione causale con gli stati . . . 47
8.4 Il modello finale . . . 48
8.5 Dall'hardware embedded all'applicazione Android . . . 49

9 Sperimentazione e selezione dei parametri del modello 50
9.1 Metodologia . . . 50
9.2 Ampiezza della finestra di analisi . . . 52
9.3 Profondità della rete . . . 53
9.4 Numero di blocchi temporali . . . 56
9.5 Generalizzazione al mondo reale . . . 57
9.6 Scelta del formato . . . 58

10 SentoMeglio 60
10.1 Architettura generale dell'applicazione 60
10.2 Interfaccia utente 61
10.2.1 Schermata daily 61
10.2.2 Schermata developer 62
10.3 Cattura e riproduzione audio: la libreria Oboe 63
10.3.1 Configurazione delle stream 64
10.3.2 Il callback audio 64
10.4 La pipeline di elaborazione audio 64
10.4.1 Il flusso tra i due thread 65
10.4.2 Estrazione della magnitudine 66
10.4.3 Maschera e risintesi 67
10.5 Strategie adottate per minimazzare la latenza 67
10.5.1 Latenza hardware 68
10.5.2 Latenza algoritmica 68
10.5.3 Thread e code 69
10.5.4 RTF ed EMA 70
10.6 Esecuzione in background 70
10.6.1 Android Foreground Service 70

11 Conclusioni 72
11.1 Limiti del lavoro 72
11.2 Sviluppi futuri 72

Acronimi 74

Bibliografia 78

Elenco delle tabelle

5.1 Costo delle convoluzioni impiegate nel ResBlock 25
5.2 Contributo di ciascuna convoluzione dilatata al *receptive field* complessivo. 29
5.3 Tabella di riferimento del modello baseline STFT-TCNN [16], nei formati *float32* e quantizzato *int8*, valutato sul dataset Valentini (VoiceBank-DEMAND). 29

6.1 Metriche oggettive (formato float32) sul dataset Valentini. 34
6.2 Metriche oggettive (formato quantizzato QDQ int8) sul dataset Valentini. 35
6.3 Metriche soggettive DNSMOS (formato float32) stimate su due clip audio 35

7.1 Confronto architetturale tra il baseline STFT-TCNN e ERB-TCNN. 42
7.2 Confronto delle metriche di training tra modello baseline e il modello che implementa il modulo ERB. 43
7.3 Occupazione di risorse e latenza di inferenza dei due modelli in formato quantizzato *int8* sul target STM32N6. 44

9.1 Metriche oggettive dell'esperimento che varia l'ampiezza della finestra 52
9.2 Punteggio percettivo complessivo DNSMOS al variare dell'ampiezza della finestra. 53
9.3 Metriche oggettive per l'esperimento che vaaria il numero di livelli per blocco. 54
9.4 Punteggio percettivo complessivo DNSMOS al variare del numero di livelli. 55
9.5 Costo su dispositivo al variare del numero di livelli. 55
9.6 Metriche oggettive al variare del numero di blocchi TCN. 56
9.7 Punteggio percettivo complessivo DNSMOS al variare del numero di blocchi. 57
9.8 Costo su dispositivo al variare del numero di blocchi. 57

9.9 Metriche oggettive del modello riaddestrato sul dataset DNS Challenge: non direttamente confrontabili con le tabelle precedenti, ottenute su un test set diverso e più semplice. 58
9.10 Comparazione metriche soggettive tra il modello addestrato sul solo dataset iniziale e quello addestrato su DNS Challenge, sulle due clip reali di test (formato *float32*). 58

Elenco delle figure

1.1 Ecosistema applicativo di SentoMeglio: il microfono dello smartphone cattura la voce dell'interlocutore mescolata al rumore ambientale, la rete neurale esegue l'inferenza direttamente sul dispositivo, e il segnale ripulito viene trasmesso all'auricolare. 3

2.1 Struttura gerarchica delle tecniche di Intelligenza Artificiale. Ogni area contiene una sua specializzazione. 6
2.2 Architettura di una Rete Neurale Ricorrente (RNN): a differenza della rete feedforward, ogni neurone dello strato nascosto riceve in ingresso anche il proprio output al passo temporale precedente. . . . 8
2.3 Confronto tra la ReLU e le varianti proposte per attenuare il problema della *dying ReLU*. 9
2.4 Architettura di una Rete Neurale Ricorrente (RNN). 11

4.1 Confronto tra lo spettrogramma di un segnale rumoroso (sopra) e il corrispondente segnale *denoised* (sotto). 20

5.1 Pipeline di elaborazione della STFT-TCNN. 22
5.2 Struttura interna del blocco residuo (ResBlock): la convoluzione pointwise iniziale espande i canali, la D-Conv dilatata elabora il contesto temporale, e la pointwise finale li riporta alla dimensione di ingresso; la connessione residua somma l'ingresso originale all'uscita del ramo convoluzionale. 24
5.3 Convoluzione 1D ordinaria. 26
5.4 Convoluzione pointwise 1×1. 26
5.5 Convoluzione depthwise-separable. 27
5.6 Schema dell'architettura della TCNN utilizzata nel modello STFT-TCNN. 27

6.1 Curva isofonica a 40 phon (ISO 226:2023). 31
6.2 Costruzione delle sottobande sull'asse delle frequenze. I centri c_i sono equispaziati in scala Mel. 33

6.3 Andamento dell'addestramento per le varianti $\mathcal{L}_{loud}$ e $\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$. 36

7.1 Approssimazione ERB del filtro uditivo: il rettangolo ha la stessa altezza di picco e la stessa area del filtro reale, e la sua larghezza definisce la banda critica. 38
7.2 Collocazione del modulo ERB nell'architettura ERB-TCNN. 40
7.3 Schema qualitativo del banco di filtri del modulo ERB. 41

8.1 Confronto tra convoluzione causale e non causale su un singolo layer dilatato ($K = 3$, $d = 2$): nel pannello non causale (sopra) l'uscita dipende anche da un campione futuro $t + d$, mentre nel pannello causale (sotto) dipende solo da campioni presenti e passati. 46

10.1 Architettura dell'applicazione ad alto livello: il livello Kotlin (interfaccia utente) e quello C++ (acquisizione audio, DSP e inferenza) comunicano attraverso l'interfaccia JNI. 61
10.2 Schermata *daily*. 62
10.3 Schermata *developer*. 63
10.4 Percorso del segnale dal microfono alla riproduzione: agli estremi le fasi eseguite dal thread audio, al centro quelle del thread di inferenza, disaccoppiati da code *lock-free*. 65

Capitolo 1

Introduzione

1.1 Ipoacusia

L'ipoacusia consiste nella riduzione o nella perdita della capacità uditiva. Può interessare un solo orecchio o entrambi, con gradi di severità che spaziano dal lieve al profondo, con effetti rilevanti sulla qualità della vita e sulla capacità di interazione sociale. Le tecnologie per il trattamento dell'ipoacusia esistono, ma rimangono spesso inaccessibili a causa dei costi elevati degli apparecchi acustici moderni.

Contrariamente a quanto si possa pensare, la sfida principale per chi è affetto da ipoacusia non è il silenzio, bensì il "caos". Se in ambienti silenziosi la comunicazione può risultare agevole, la presenza di rumore di fondo — come il brusio di un ristorante affollato o il frastuono del traffico — compromette drasticamente la comprensione. In queste condizioni, un sistema uditivo danneggiato fatica a separare la voce dell'interlocutore dai disturbi circostanti. Questo fenomeno è noto in letteratura come il "problema del cocktail party": l'incapacità del sistema uditivo di isolare un segnale d'interesse all'interno di una sovrapposizione di stimoli acustici differenti.

1.2 Apparecchi acustici

Gli apparecchi acustici (*hearing aids*) sono dispositivi elettronici miniaturizzati il cui funzionamento storico si è basato sulla semplice amplificazione: ricevono il segnale acustico e lo riemettono a volume aumentato.

Strutturalmente integrano microfono, processore digitale, amplificatore, ricevitore e batteria. Il microfono converte le onde sonore in segnale digitale, il processore lo elabora, l'amplificatore ne incrementa l'intensità, e il ricevitore converte gli impulsi elettrici in pressione acustica.

A partire dagli anni '90, sono stati introdotti metodi di elaborazione del segnale come la *spectral subtraction* per sottrarre una stima dello spettro del rumore, il filtraggio adattativo per modulare il guadagno tra bande di frequenza e il filtro di Wiener per l'ottimizzazione statistica del segnale. Tali approcci, tuttavia, operano sotto il presupposto che il rumore sia stazionario [1]. Negli scenari reali, le sorgenti di rumore sono non stazionarie, con variazioni significative di intensità e caratteristiche temporali. La *spectral subtraction* migliora il comfort percettivo ma non aumenta la comprensione del parlato così come il filtraggio adattativo e il filtro di Wiener, sebbene utili per la riduzione del rumore di fondo, rimangono confinati a vincoli hardware severi e ad assunzioni statistiche ideali. Di conseguenza, l'efficacia degli apparecchi acustici tradizionali ha raggiunto un plateau negli ultimi decenni [2], con miglioramenti marginali tra generazioni successive, senza risolvere il problema fondamentale della separazione del parlato in ambienti ad alta complessità acustica.

1.3 Cambio di paradigma

A fronte dei limiti dei metodi tradizionali, questa tesi propone un approccio basato su reti neurali profonde per lo *speech enhancement* in tempo reale su smartphone. Come illustrato nella Figura 1.1, l'obiettivo è isolare la voce dell'interlocutore sopprimendo il rumore ambientale. L'architettura si basa su un'applicazione Android che acquisisce l'audio, esegue l'inferenza in locale con un modello neurale leggero e trasmette il segnale elaborato all'auricolare con latenza minima. Lo smartphone fornisce la potenza di calcolo necessaria, senza richiedere hardware dedicato aggiuntivo.

Questo scenario impone vincoli stringenti e parzialmente conflittuali tra loro: bassa latenza algoritmica, consumo energetico contenuto, utilizzo limitato di RAM e carico di calcolo compatibile con l'elaborazione in tempo reale. Questi requisiti, analizzati in dettaglio nel Capitolo 3, sono spesso incompatibili con le architetture neurali standard, particolarmente esigenti in termini di risorse e di memoria.

La scelta di puntare su un telefono di uso comune, invece che su un dispositivo dedicato, non nasce solo da un'esigenza tecnica ma da una precisa presa di posizione: le soluzioni presenti nel mercato hanno un prezzo che spesso li rende fuori portata per moltissime persone che ne avrebbero bisogno, mentre il cellulare è ormai un oggetto che quasi chiunque possiede già. L'obiettivo di questo lavoro è quindi duplice: dimostrare che un sistema di riduzione del rumore basato su reti neurali può funzionare in tempo reale su hardware di consumo, e gettare le basi per rendere questo tipo di tecnologia economicamente sostenibile per chi convive con l'ipoacusia.

Dal punto di vista sperimentale, la tesi fornisce due contributi principali: la progettazione di un'architettura leggera a bassa latenza e l'implementazione dell'infrastruttura Android per l'inferenza in tempo reale.

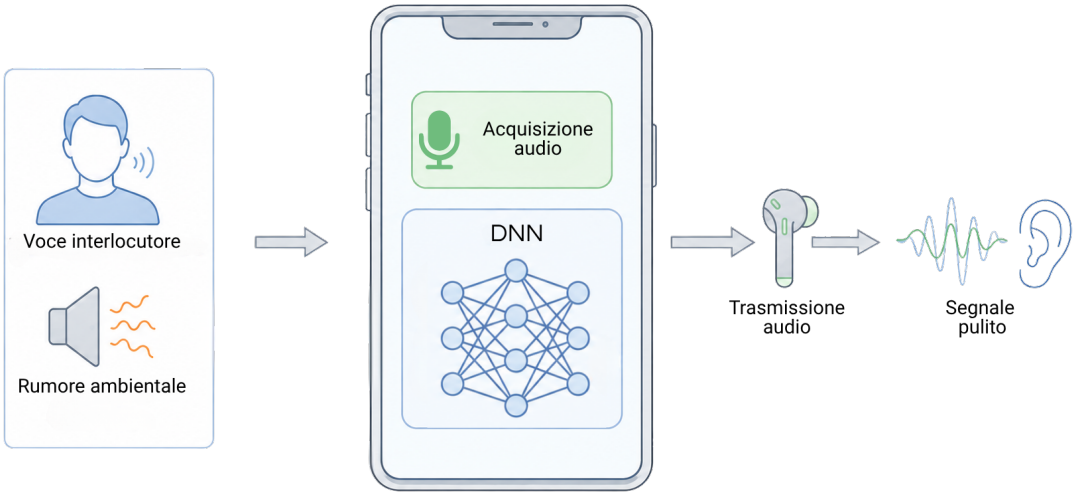


Figura 1.1: Ecosistema applicativo di SentoMeglio: il microfono dello smartphone cattura la voce dell'interlocutore mescolata al rumore ambientale, la rete neurale esegue l'inferenza direttamente sul dispositivo, e il segnale ripulito viene trasmesso all'auricolare.

1.4 Struttura della tesi

Il resto del documento è organizzato seguendo il percorso che ha portato dalla teoria all'applicazione funzionante. Il Capitolo 2 richiama i concetti di base delle reti neurali necessari a comprendere le scelte progettuali successive, mentre il Capitolo 3 inquadra il problema dello *speech enhancement* e i vincoli imposti dall'esecuzione su dispositivo mobile. Il Capitolo 4 introduce la rappresentazione del segnale nel dominio della frequenza su cui si basa l'intera pipeline.

I capitoli centrali descrivono l'architettura proposta e le sue componenti: il Capitolo 5 presenta il modello STFT-TCNN, il Capitolo 6 discute la funzione di costo Loud-loss ispirata a criteri psicoacustici, il Capitolo 7 approfondisce il modulo ERB per la compressione delle bande in frequenza e il Capitolo 8 affronta gli adattamenti richiesti per il funzionamento in streaming. Il Capitolo 9 raccoglie la sperimentazione condotta per selezionare i parametri del modello finale, confrontando le diverse configurazioni testate.

Gli ultimi capitoli chiudono il cerchio tra ricerca e prodotto: il Capitolo 10 descrive SentoMeglio, l'applicazione Android in cui il modello è stato integrato, e il Capitolo 11 riprende i risultati ottenuti e discute i possibili sviluppi futuri del

lavoro.

Capitolo 2

Reti neurali

Prima di descrivere l'architettura del modello sviluppato in questo lavoro, è utile fissare la terminologia e il quadro concettuale di riferimento. Meedeniya [3] distingue tre livelli di astrazione spesso confusi tra loro e schematizzati in Figura 2.1: l'**IA** è il campo che mira a costruire sistemi in grado di emulare i comportamenti umani e quindi di svolgere compiti che richiederebbero normalmente il loro intervento. Il **ML** ne è una specializzazione, anziché programmare regole esplicitamente, questi sistemi apprendono le proprie regole dai dati tramite addestramento su dataset. Il **DL** invece utilizza reti composte da molti strati fitti che permettono di estrarre progressivamente caratteristiche di livello crescente dai dati grezzi ed è proprio questa profondità che gli consente di modellare relazioni complesse come quelle presenti nel segnale audio.

Il DL ha raggiunto risultati pratici rilevanti negli ultimi anni per effetto di due fattori concomitanti: la disponibilità di grandi dataset e l'aumento della potenza di calcolo delle GPU. Applicazioni come il riconoscimento di immagini, il riconoscimento vocale e l'elaborazione del linguaggio naturale ne hanno beneficiato direttamente.

2.1 Tipologie di apprendimento

La natura dei dati disponibili determina quale approccio all'apprendimento usare. Gli algoritmi di deep learning si dividono in due categorie principali:

- L'**apprendimento supervisionato** (*supervised learning*) utilizza dataset annotati: ogni input ha un'etichetta corrispondente (*label* o *ground truth*) che rappresenta la risposta corretta. La rete neurale impara confrontando le sue predizioni con le etichette, correggendo gradualmente i propri errori fino a sviluppare una mappatura affidabile tra input e output.

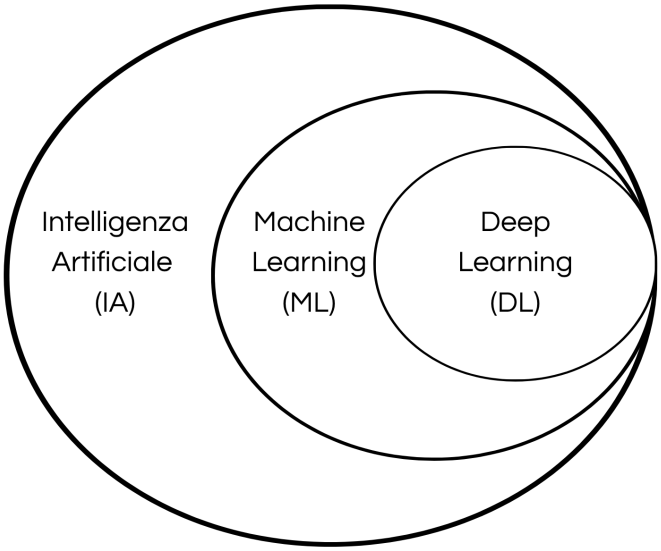


Figura 2.1: Struttura gerarchica delle tecniche di Intelligenza Artificiale. Ogni area contiene una sua specializzazione.

- L'**apprendimento non supervisionato** (*unsupervised learning*) lavora su dati senza etichette. L'algoritmo impara autonomamente pattern, strutture o raggruppamenti (*clustering*) nascosti nei dati, senza indicazioni esterne sulla soluzione.

In questa tesi, lo *speech enhancement* è trattato come un compito supervisionato: il modello viene addestrato su coppie di segnali audio rumorosi e puliti e impara quindi a stimare la voce per parlante a partire da un audio degradato. Il dataset e i dettagli di addestramento sono descritti nei capitoli successivi.

2.2 Le Reti Neurali Artificiali

Le RNA traggono ispirazione dal sistema nervoso umano, replicando il modo in cui i neuroni biologici elaborano e trasmettono segnali. Essi sono composti da neuroni artificiali interconnessi, organizzati in strati sequenziali dove ogni neurone riceve un segnale di input, lo elabora applicando una funzione di attivazione non lineare e produce un output che alimenta quelli dello strato successivo.

Una RNA[4] standard è costruita attorno a tre tipologie principali di strati che processano i dati in sequenza, vedi Figura 2.2:

1. **Input Layer:** Riceve i dati grezzi del dominio di interesse (pixel di immagini, campioni audio, sequenze temporali). Ogni neurone corrisponde a una caratteristica o a un campione del segnale. Non avviene elaborazione in questo strato, è semplicemente il punto di ingresso attraverso il quale i dati entrano nella rete.

2. **Hidden Layers:** Sono gli strati intermedi dove avviene la trasformazione e l'estrazione progressiva di caratteristiche. Ogni neurone di uno strato nascosto riceve i segnali dai neuroni dello strato precedente, applica una somma pesata dei segnali di input e poi passa il risultato attraverso una funzione di attivazione non lineare. La profondità della rete — determinata dal numero di tali strati — governa la capacità della rete di modellare relazioni complesse nei dati.

3. **Output Layer:** Produce il risultato finale dell'elaborazione: predizioni, classificazioni o segnali elaborati. Nel contesto dello *speech enhancement*, l'output layer genera il segnale vocale ripulito dal rumore.

Ciascuna connessione tra i neuroni è caratterizzata da un peso (w) che determina l'importanza relativa del segnale in ingresso. L'addestramento aggiorna questi pesi iterativamente mediante *backpropagation*, minimizzando una funzione di perdita che misura lo scarto tra le predizioni della rete e i valori di *ground truth*.

2.2.1 Funzioni di attivazione

Dopo la somma pesata degli ingressi, ciascun neurone applica un'ultima trasformazione, la **funzione di attivazione**, che introduce la non linearità appena menzionata. Senza di essa l'intera rete, per quanti strati venga impilata, si riduce matematicamente a un'unica trasformazione lineare dell'ingresso, per definizione incapace di rappresentare le relazioni non lineari che nella maggior parte dei problemi reali legano input e output. La scelta di questa funzione non è univoca e ha attraversato fasi diverse nella storia del DL.

Le prime reti si affidavano principalmente a funzioni **saturanti**, tra cui la sigmoide $\sigma(x) = \frac{1}{1+e^{-x}}$ e la tangente iperbolica $\tanh(x)$, che comprimono qualunque

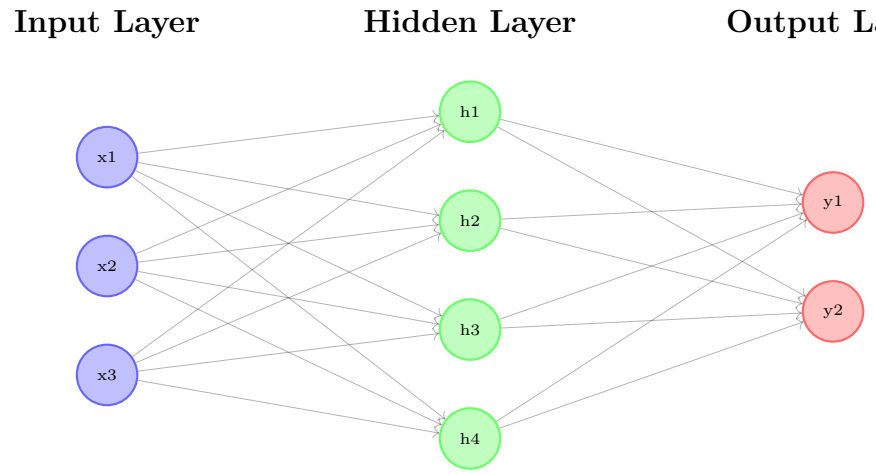


Figura 2.2: Architettura di una Rete Neurale Ricorrente (RNN): a differenza della rete feedforward, ogni neurone dello strato nascosto riceve in ingresso anche il proprio output al passo temporale precedente.

ingresso rispettivamente nell'intervallo (0,1) e (−1,1). Proprio questa proprietà le rende ancora oggi utili quando l'uscita della rete deve rappresentare una probabilità o un fattore di scala limitato, come accade per la maschera spettrale finale del modulo ERB (Capitolo 7). Negli strati intermedi di una rete profonda, però, il loro impiego sistematico si scontra con un limite: per ingressi grandi in valore assoluto la curva si appiattisce quasi del tutto, il gradiente locale si assottiglia e, propagandosi a ritroso strato dopo strato, può ridursi fino a svanire, rendendo l'addestramento dei livelli più profondi estremamente lento.

La **ReLU**, definita come $f(x) = \max(0, x)$, ha in gran parte soppiantato le funzioni saturanti negli strati nascosti delle reti moderne. Il calcolo che richiede è minimo e, per ogni ingresso positivo, il gradiente resta costante indipendentemente da quanto grande sia il valore attraversato, il che rende l'addestramento più stabile anche su reti profonde. Ha però una controindicazione nota come problema della *dying ReLU*: se durante l'addestramento i pesi in ingresso a un neurone lo spingono in una regione in cui l'uscita è sempre negativa, il gradiente locale si azzera, il neurone smette di ricevere correzioni e resta di fatto inattivo per il resto del training, un fenomeno concettualmente distinto dal problema della scomparsa del gradiente, ma dagli effetti altrettanto dannosi sull'apprendimento.

Per contenere questo rischio sono state proposte alcune varianti che, a differenza della ReLU, assegnano un'uscita diversa da zero anche alla parte negativa dell'ingresso, confrontate in Figura 2.3. La **Leaky ReLU** sostituisce il tratto piatto con una retta di pendenza α piccola, fissata a priori e mai modificata durante l'addestramento, così che un gradiente residuo, per quanto minimo, continui comunque a

fluire quando l'ingresso è negativo. La **PReLU** riprende la stessa forma funzionale, ma tratta α come un parametro allenabile, lasciando che sia l'addestramento a determinare quanto debba pesare la parte negativa. La **ELU** adotta invece per gli ingressi negativi una curva esponenziale che si stabilizza asintoticamente a $-\alpha$, evitando lo spigolo nell'origine presente nelle altre varianti.

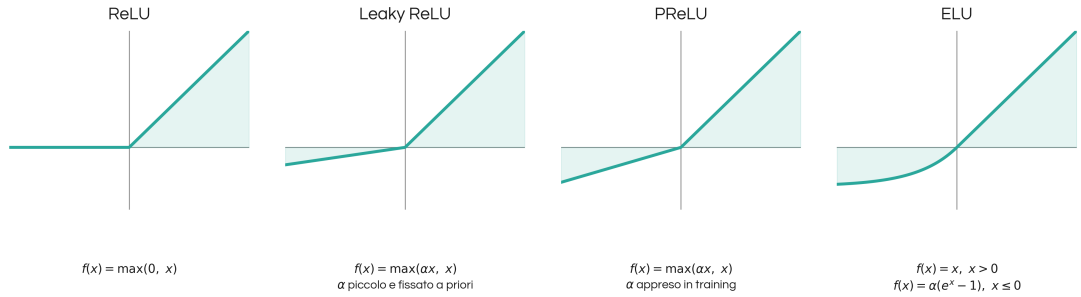


Figura 2.3: Confronto tra la ReLU e le varianti proposte per attenuare il problema della *dying ReLU*.

2.2.2 Il ciclo di addestramento

L'addestramento corregge i pesi della rete tramite backpropagation, minimizzando una funzione di perdita, vale la pena ripercorrere per intero questo meccanismo, dato che ne costituisce il fondamento concettuale, valido a prescindere dal compito specifico o dall'architettura scelta.

Ogni ciclo di addestramento, detto **epoca**, fa attraversare alla rete l'intero insieme di addestramento e si articola in due fasi complementari. Nella prima, il *forward pass*, un esempio in ingresso percorre la rete strato dopo strato fino a produrre un'uscita, che viene confrontata con il valore desiderato tramite la funzione di perdita descritta poco sopra: quanto più la predizione si allontana dal target, tanto più alto è il valore restituito. Per motivi di efficienza gli esempi non vengono presentati uno a uno né tutti simultaneamente, ma raggruppati in piccoli lotti (*batch*), un compromesso tra la stabilità della stima della perdita e il costo computazionale di ogni passo.

Nella seconda fase, il *backward pass*, la **backpropagation** applica la regola della catena per calcolare, strato dopo strato e a ritroso rispetto al forward pass, il gradiente della perdita rispetto a ciascun peso della rete: una quantità che indica in che direzione, e con quale intensità relativa, ciascun peso dovrebbe muoversi per ridurre l'errore corrente. Un algoritmo di ottimizzazione, l'*optimizer*, traduce questa informazione in un aggiornamento effettivo, spostando ogni peso nella direzione opposta al proprio gradiente di una quantità regolata dal *learning rate*: un passo troppo ampio rischia di superare il minimo della perdita a ogni iterazione, uno

troppo piccolo rende l'addestramento estremamente lento. Gli optimizer più diffusi, tra cui Adam, adattano questo passo separatamente per ciascun peso sulla base dell'andamento recente del suo stesso gradiente, piuttosto che applicare a tutti un identico fattore fisso. Il learning rate, inoltre, non deve necessariamente restare fisso per l'intera durata dell'addestramento: è pratica comune ridurlo quando la perdita di validazione smette di migliorare, così da affinare la soluzione con passi via via più piccoli man mano che ci si avvicina a un minimo, una strategia nota come *scheduling* del learning rate.

Ripetendo questo ciclo su molte epoche la perdita tende a diminuire, ma un valore basso calcolato sugli stessi dati impiegati per correggere i pesi non garantisce che la rete si comporti altrettanto bene su dati mai visti: per questo una porzione dell'insieme disponibile, l'insieme di **validazione**, viene tenuta sistematicamente fuori dall'aggiornamento dei pesi e usata soltanto per misurare, epoca dopo epoca, la capacità di **generalizzazione** della rete. Il confronto tra le due misure permette di riconoscere due esiti indesiderati e opposti. Si parla di **underfitting** quando la rete non riesce a ridurre a sufficienza la perdita neanche sui dati di addestramento, segno che la sua capacità o la durata dell'addestramento non bastano a cogliere la struttura del problema. Si parla invece di **overfitting** quando la perdita di addestramento continua a scendere mentre quella di validazione ricomincia a salire: la rete, in questo caso, sta memorizzando le particolarità del proprio insieme di addestramento, incluso il suo rumore, invece di apprendere una regolarità che si estenda a esempi nuovi. È proprio per intercettare questo secondo scenario che si ricorre spesso a un criterio d'arresto anticipato (*early stopping*): l'addestramento viene interrotto quando la perdita di validazione smette di migliorare per un numero di epoche consecutive, conservando i pesi del punto migliore già raggiunto invece di proseguire fino all'ultima epoca prevista.

2.2.3 Architetture principali di reti neurali

Non tutte le RNA sono uguali: la topologia della rete — come i neuroni sono interconnessi — determina quali tipi di dati e di problemi può gestire efficacemente. Le tre principali architetture sono:

- **Feedforward Neural Networks (FNN):** Sono le reti più semplici, dove i dati fluiscono in una sola direzione, dal layer di input attraverso i layer nascosti fino all'output layer. Non esistono connessioni ricorrenti (*feedback loops*). Le FNN sono efficaci per compiti su dati statici e indipendenti, come la classificazione di immagini singole o la previsione su dati tabulari. La loro semplicità le rende computazionalmente efficienti, ma questa loro caratteristica le limita nel catturare dipendenze temporali.

- **Recurrent Neural Networks (RNN):** Introducono connessioni ricorrenti che permettono a ciascun neurone di ricevere feedback dal proprio output e da quelli degli altri neuroni dello stesso strato, calcolati al passo temporale precedente (vedi Figura 2.4). Questa architettura è ideale per dati sequenziali e temporali, dove l'ordine e il contesto storico sono critici: testo, serie temporali, audio. Gli RNN mantengono uno "stato nascosto" che funge da memoria a breve termine, permettendo alla rete di integrare informazioni passate nelle decisioni presenti. Tuttavia, gli RNN standard soffrono del problema della scomparsa o esplosione del gradiente [5] durante l'addestramento. Varianti come *Long Short-Term Memory* e *Gated Recurrent Unit* risolvono questo problema introducendo meccanismi di gating che controllano il flusso di informazioni attraverso il tempo.

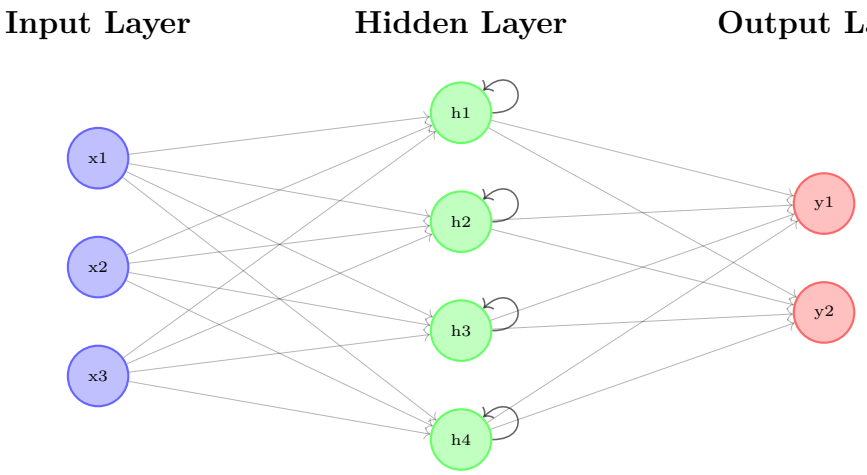


Figura 2.4: Architettura di una Rete Neurale Ricorrente (RNN).

- **Convolutional Neural Networks (CNN):** Sfruttano la struttura spaziale dei dati applicando filtri (*kernel*) che scorrono sul segnale di input. I CNN sono particolarmente efficaci per l'elaborazione di segnali multi-dimensionali: immagini, audio e video. Gli strati convoluzionali estraggono feature locali (spigoli, pattern, motivi), mentre gli strati di pooling riducono la dimensionalità comprimendo l'informazione. Nel dominio dell'audio, le CNN catturano pattern frequenziali e temporali locali, rendendole adatte a compiti come lo *speech enhancement* e il riconoscimento vocale.

La combinazione di più strati e la scelta dell'architettura sono il cuore del *Deep Learning*: più strati permettono di apprendere rappresentazioni di livello crescente, dai pattern semplici alle strutture astratte a partire dai dati grezzi.

Capitolo 3

Speech Enhancement

Lo *speech enhancement* raccoglie le tecniche di elaborazione del segnale che mirano a migliorare la qualità percettiva e l'intelligibilità del parlato. L'obiettivo è ricostruire un segnale vocale quanto più possibile privo di disturbi a partire da una miscela acustica corrotta, in cui la voce di interesse è contaminata da rumore di fondo, rumore dei sensori e riverberazione ambientale [6].

Il segnale osservato può essere modellato come la somma di una componente vocale desiderata e di una o più componenti indesiderate:

$$x(t) = s(t) + n(t) \tag{3.1}$$

dove:

- $x(t)$: segnale rumoroso osservato
- $s(t)$: audio pulito
- $n(t)$: rumore additivo

Il modello dell'Eq. (3.1) è additivo e a singolo canale: descrive bene il rumore di fondo e quello introdotto dai sensori di acquisizione, ma non la riverberazione, che agisce moltiplicando il segnale per la risposta impulsiva dell'ambiente e non è quindi rappresentabile con un semplice termine di somma.

Questa *task* dunque mira a separare, attenuare o compensare tali componenti di disturbo, preservando le caratteristiche rilevanti della voce umana. La natura del rumore $n(t)$ influisce sulla complessità del problema. Si distinguono tipicamente:

- **Rumore stazionario**: caratterizzato da proprietà statistiche costanti nel tempo (es. il rumore bianco). Questi segnali sono più facilmente modellabili e rimovibili.

- **Rumore non stazionario**: include segnali con variazioni rapide e imprevedibili, come il brusio di fondo in un locale affollato (*babble noise*), colpi di tosse o il traffico stradale. La soppressione di tali disturbi è la sfida principale per i modelli moderni, poiché le componenti spettrali si sovrappongono frequentemente a quelle del parlato desiderato.

Un'attenuazione eccessiva del rumore può introdurre artefatti percepibili e rendere il segnale innaturale, quasi tutti i metodi di elaborazione alterano almeno in parte anche la componente vocale desiderata. Lo *speech enhancement* è quindi un problema di compromesso: riduzione del rumore da un lato, preservazione della naturalezza e dell'intelligibilità del parlato dall'altro [7].

3.1 Masking e mapping

Definito il problema, resta da stabilire cosa, esattamente, un sistema debba restituire in uscita per risolverlo. Al di là dei dettagli implementativi, la letteratura sullo *speech enhancement* supervisionato distingue a questo proposito due paradigmi di stima, noti come **mapping** e **masking** [8], che si differenziano per il legame, più o meno vincolato, tra la grandezza osservata in ingresso e quella stimata in uscita.

- Nel **magnitude mapping**, la rete stima direttamente il valore assoluto della magnitudine pulita $|\hat{S}(m,k)|$ a partire da quella rumorosa $|X(m,k)|$, trattando il problema come una regressione priva di vincoli espliciti tra ingresso e uscita.
- Nel **magnitude masking**, la rete non stima la magnitudine pulita in valore assoluto, ma una maschera $\mathcal{M}(m,k)$, un fattore di correzione reale e limitato a un intervallo ristretto di valori, che va poi applicato per moltiplicazione a quella rumorosa per ottenerne la stima. Il riferimento più citato in letteratura è la *ideal ratio mask* (IRM), il target ideale a cui la maschera stimata dalla rete tende quanto più l'addestramento riesce bene: esprime il rapporto tra l'energia della componente vocale e quella della miscela rumorosa, ed è per costruzione compresa nell'intervallo [0,1].

La differenza tra i due approcci non è solo formale, ma incide direttamente sulla difficoltà del problema di apprendimento. Il masking richiede alla rete di stimare una correzione relativa a un ingresso che, soprattutto nelle bande e negli istanti a rapporto segnale-rumore favorevole, contiene già buona parte dell'informazione corretta: il problema risulta così più vincolato e, a parità di dati, più semplice da addestrare in modo stabile. Il mapping diretto, al contrario, deve ricostruire da zero un valore di magnitudine il cui intervallo di ampiezze dipende dal livello di registrazione del segnale ed è perciò enormemente più variabile, il che rende la regressione meno vincolata e più esposta a un addestramento instabile.

Anche per questo, oltre che per l'evidenza empirica che i target basati su maschera tendono a preservare meglio l'intelligibilità del parlato rispetto al mapping diretto [8], il masking è l'approccio scelto dalla maggior parte dei sistemi neurali pensati per il tempo reale su dispositivi a risorse limitate, ed è anche la strada seguita dal modello sviluppato in questa tesi. Il mapping diretto resta comunque preferibile quando il compito non si limita ad attenuare un disturbo sovrapposto al parlato, ma richiede di ricostruire un contenuto del tutto assente dal segnale osservato.

Resta infine un limite comune a entrambi i paradigmi, quando applicati come in questo lavoro alla sola magnitudine: la fase dello spettro stimato è sempre quella, non ripulita, del segnale rumoroso. È una scelta che semplifica la pipeline ma che pone un tetto noto alla qualità massima ricostruibile.

3.2 Efficienza computazionale e vincoli implementativi

I modelli basati su DNN hanno migliorato le prestazioni in questo ambito, ma il loro impiego nei dispositivi *mobile* introduce vincoli che non hanno equivalenti nel contesto desktop o cloud. Come anticipato nella Sezione 1.3, l'obiettivo di questo lavoro è eseguire lo *speech enhancement* in tempo reale su smartphone, trasmettendo il segnale elaborato all'auricolare con latenza minima. I quattro vincoli principali sono:

- **Latenza**: il ritardo deve essere abbastanza basso da non risultare percepibile come sfasamento tra il labiale dell'interlocutore e l'audio ricevuto;
- **Consumo energetico**: il modello deve operare in background senza compromettere l'autonomia della batteria;
- **Occupazione di memoria**: l'utilizzo di RAM deve restare entro i limiti dell'hardware dove viene eseguito, in parallelo con le altre applicazioni;
- **Complessità computazionale**: il numero di operazioni per frame deve permettere l'elaborazione in tempo reale senza surriscaldare il processore o causare interruzioni audio.

Un'elevata qualità audio è spesso associata a modelli di grandi dimensioni per questo motivo rispettare questi vincoli richiede architetture neurali leggere, con un numero limitato di parametri e un campo ricettivo controllato [9].

3.3 Metriche e valutazione dei modelli

Come introdotto nel Capitolo 2, il *task* precedentemente introdotto rientra nel paradigma dell'apprendimento supervisionato. Il modello viene addestrato su coppie di segnali, questo approccio richiede dataset estesi e rappresentativi di molteplici condizioni acustiche. Diventa praticamente impossibile coprire ogni combinazione di tipo di rumore, ambiente e interlocutore, il che può limitare la capacità di generalizzazione a scenari non osservati durante l'addestramento.

La valutazione delle prestazioni è parte integrante del processo di sviluppo. A tal fine, vengono adottate metriche oggettive standardizzate che consentono un confronto quantitativo tra metodi differenti. Le metriche più diffuse sono:

- SNR: misura il livello di attenuazione del rumore rispetto al segnale desiderato. Un tale incremento indica una maggiore riduzione del rumore.
- SI-SNR: variante dello SNR pensata per essere insensibile a un fattore di scala costante tra il segnale stimato $\hat{s}(t)$ e quello di riferimento $s(t)$. Si proietta anzitutto $\hat{s}(t)$ sulla direzione di $s(t)$ tramite un coefficiente $\alpha = \langle \hat{s}, s \rangle / \|s\|^2$, quindi si confronta l'energia di questa proiezione con quella del residuo:

  $$\text{SI-SNR} = 10 \log_{10} \frac{\|\alpha s\|^2}{\|\alpha s - \hat{s}\|^2} \tag{3.2}$$

  A differenza dello SNR ordinario, un errore sistematico di guadagno tra stima e riferimento — che non altera l'intelligibilità percepita — non viene quindi penalizzato.
- PESQ: fornisce una stima oggettiva della qualità percepita del parlato. Il punteggio assume tipicamente valori compresi tra -0.5 e 4.5, dove valori più elevati indicano una migliore qualità percepita.
- STOI: valuta l'intelligibilità del parlato stimando la correlazione tra il segnale pulito e quello processato nel dominio temporale a breve termine. Il punteggio varia tra 0 e 1, con valori più vicini a 1 indicanti maggiore intelligibilità.

Queste metriche catturano aspetti differenti delle prestazioni: un algoritmo può migliorare significativamente l'SNR ma peggiorare la qualità percepita (PESQ) o l'intelligibilità (STOI), ad esempio introducendo artefatti metallici. Accanto alle metriche oggettive, le valutazioni soggettive rivestono un ruolo ugualmente importante, essendo basate su test di ascolto condotti con partecipanti umani. Tra queste:

- MOS: punteggio medio assegnato dagli ascoltatori alla qualità complessiva del segnale, generalmente su una scala discreta (ad esempio da 1 a 5), dove valori più alti indicano qualità migliore.

- SIG: misura soggettiva della distorsione percepita nella componente vocale.
- BAK: valuta quanto il rumore di fondo risulti percepibile o intrusivo.
- OVRL: sintetizza la qualità globale percepita.

Queste costituiscono il riferimento più diretto per la qualità percepita in quanto riflettono l'esperienza dell'ascoltatore [10]. Il loro limite, tuttavia, è il costo: richiedono di organizzare sessioni di ascolto con un gruppo di partecipanti umani, un processo lento e poco pratico durante lo sviluppo iterativo di un modello.

Per la valutazione dei modelli sviluppati in questo lavoro, si utilizza il framework **DNSMOS** messo a disposizione da Microsoft nella repository ufficiale del *DNS-Challenge* [11]. Si tratta di una rete neurale addestrata a riprodurre i punteggi che assegnerebbe un gruppo di ascoltatori. Inoltre, a differenza di metriche come PESQ o STOI, non ha bisogno del segnale pulito originale come riferimento per il confronto.

Capitolo 4

Analisi nel dominio della frequenza

La maggior parte dei sistemi moderni di *speech enhancement* non opera direttamente sui campioni temporali del segnale, ma su una rappresentazione nel dominio della frequenza. Questo capitolo descrive la *Short-Time Fourier Transform* (STFT), l'operazione che permette di passare dal dominio del tempo a quello tempo-frequenza, e analizza i parametri che ne regolano il funzionamento nelle implementazioni pratiche.

4.1 Proprietà in frequenza della voce

La scelta di operare in questo dominio poggia su proprietà del segnale vocale e su evidenze sperimentali. Nossier et al. [12] hanno confrontato sistematicamente approcci nel dominio del tempo e della frequenza basati su deep learning, riscontrando una superiorità degli approcci frequenziali su metriche di qualità e intelligibilità.

- **Sparsità spettrale**: Il parlato manifesta una struttura armonica ben definita — formanti e armonici del pitch — che concentra l'energia in bande specifiche e relativamente strette. Il rumore, al contrario, tende a distribuire la propria energia su tutto lo spettro o in bande distinte da quelle vocali. Nel dominio temporale, parlato e rumore si sovrappongono in ogni campione mentre in quello della frequenza, questa separazione diventa esplicita e il filtraggio può essere applicato selettivamente banda per banda [12].
- **Allineamento con la percezione uditiva**: La coclea scompone il suono per bande di frequenza, in modo analogo a una trasformata di Fourier biologica.

Le reti neurali che processano rappresentazioni spettrali apprendono caratteristiche coerenti con la percezione uditiva, il che facilita l'ottimizzazione verso le metriche di qualità percettiva descritte nel Capitolo 3 [12].

- **Quasi-stazionarietà e finestra di analisi**: Le proprietà del segnale vocale variano durante la produzione del parlato, ma su intervalli sufficientemente brevi si mantengono approssimativamente costanti. È questa condizione di quasi-stazionarietà locale che rende applicabile l'analisi di Fourier: la STFT la sfrutta suddividendo il segnale in brevi segmenti (*frame*) e calcolando la trasformata separatamente su ciascuno di essi. La durata della finestra di analisi deve tuttavia bilanciare due esigenze opposte. Da un lato, deve coprire un numero di periodi del segnale sufficiente a stimare le componenti armoniche in modo affidabile: una finestra troppo corta produce una stima spettrale instabile, in cui gli armonici si confondono e non possono essere separati. Dall'altro, non deve eccedere l'intervallo in cui vale l'ipotesi di stazionarietà locale: estendendosi oltre, abbraccia transizioni fonetiche distinte e ne mescola i contributi spettrali (*smearing*), confondendo suoni diversi in un'unica stima e degradando la qualità dell'analisi. Paliwal e Wojcicki [13] hanno studiato empiricamente questo compromesso e hanno identificato un intervallo ottimale compreso tra **15 e 35 ms**. Il sistema sviluppato in questo lavoro adotta, come verrà motivato sperimentalmente nel Capitolo 9, una finestra di 20 ms, che ricade all'interno di questo intervallo.

4.2 La Short-Time Fourier Transform (STFT)

La STFT di un segnale discreto $x[n]$ si ottiene moltiplicando il segnale per una funzione finestra $w[n]$ di durata finita e calcolando la trasformata di Fourier su segmenti successivi del segnale. Questo approccio, formalizzato nel lavoro di Allen [14] sull'analisi spettrale a breve termine tramite DFT, produce una rappresentazione tempo-frequenza del segnale.

$$X(m,k) = \sum_{n=0}^{N-1} x[n+mH]w[n]e^{-j\frac{2\pi}{N}kn} \tag{4.1}$$

dove:

- m è l'indice del **frame** temporale;
- k è l'indice del **bin** di frequenza ($k = 0, \ldots, N-1$);
- $w[n]$ è la funzione finestra (es. Hamming, Hann, ...);
- H è l'ampiezza del passo;

- N è la dimensione della trasformata di Fourier.

In generale, la finestra di analisi può avere una lunghezza L diversa da N: quando $L < N$, i campioni mancanti vengono completati con zeri (*zero-padding*), il che aumenta la risoluzione in frequenza dello spettro senza ampliare la finestra temporale. Nel sistema sviluppato in questo lavoro, $L = 320$ campioni e $N = 512$: viene applicata questa tecnica alla finestra per ottenere 257 bin di frequenza uniformemente distribuiti da 0 a 8 kHz.

4.3 Lo spettrogramma

Lo spettrogramma è la rappresentazione visiva della potenza spettrale nel tempo, ottenuta dal quadrato della magnitudine della STFT:

$$P(m, k) = |X(m, k)|^2 \tag{4.2}$$

L'asse orizzontale rappresenta il tempo, quello verticale la frequenza, mentre l'intensità cromatica in ogni cella indica la concentrazione di energia espressa in scala logaritmica in decibel. Di conseguenza, per ogni istante temporale è possibile determinare l'intensità energetica di ciascuna specifica frequenza.

La Figura 4.1 confronta lo spettrogramma di un segnale rumoroso con il corrispondente segnale pulito. La struttura armonica del parlato si concentra in diverse zone dello spettro frequenziale: le vocali si distribuiscono principalmente sulle basse frequenze, mentre le consonanti — tra cui le fricative e le occlusive — si estendono verso quelle alte. L'obiettivo del modello è produrre una maschera che attenui le componenti di disturbo preservando quelle del parlato; la natura non stazionaria del rumore circostante costituisce la sfida principale di questo processo, come descritto nel Capitolo 3.

4.4 Parametri della STFT

In un'implementazione pratica, la trasformata è definita da quattro parametri fondamentali, ognuno dei quali influenza la risoluzione della rappresentazione tempo-frequenza e la latenza algoritmica del sistema.

- **`win_length`** determina quanti campioni del segnale vengono analizzati per ciascun frame. Come discusso nella sezione precedente, la sua scelta è vincolata all'intervallo ottimale di quasi-stazionarietà del parlato (15–35 ms). Rappresenta il compromesso fondamentale tra risoluzione temporale e risoluzione in frequenza: una finestra lunga migliora la risoluzione in frequenza ma riduce quella temporale, e viceversa.

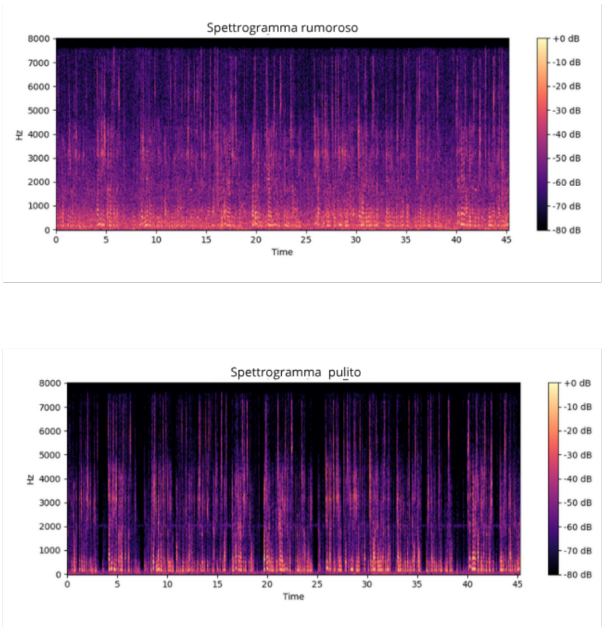


Figura 4.1: Confronto tra lo spettrogramma di un segnale rumoroso (sopra) e il corrispondente segnale *denoised* (sotto).

- **`n_fft`** determina quanti punti vengono usati per il calcolo della trasformata di Fourier. Se `n_fft > win_length`, al frame viene applicato lo *zero-padding* prima del calcolo. Il numero di bin utili è $\frac{\texttt{n_fft}}{2} + 1$ (per segnali reali). Nella maggior parte delle implementazioni lo si sceglie pari a una potenza di due per ottimizzare e velocizzare il calcolo della FFT.

- **`hop_length`** determina di quanti campioni si sposta la finestra tra un frame e il successivo. Definisce la risoluzione temporale della rappresentazione: un hop piccolo produce frame più frequenti quindi una maggiore risoluzione a discapito di più ridondanza e volume di dati. La sovrapposizione tra frame consecutivi è pari a `win_length` − `hop_length` campioni.

- **`center`** definisce l'allineamento temporale della finestra di analisi. Con `center=True`, la finestra è centrata sull'istante del frame corrente: ciò richiede un *look-ahead* di $\frac{\texttt{win_length}}{2}$ campioni futuri e l'aggiunta di padding ai bordi del segnale, introducendo una latenza algoritmica. Con `center=False`, la finestra allinea il proprio inizio al frame corrente coprendo esattamente i campioni del buffer disponibile senza applicare padding; l'assenza di *look-ahead* minimizza la latenza e consente un'elaborazione causale su flussi audio in tempo reale.

Capitolo 5

STFT-TCNN

Negli ultimi anni, le *Temporal Convolutional Neural Networks* (TCNN) si sono diffuse come alternativa ai modelli ricorrenti per l'elaborazione di segnali sequenziali, in particolare nello *speech enhancement*. Rispetto alle architetture ricorrenti introdotte nel Capitolo 2, le TCNN sfruttano convoluzioni dilatate per modellare dipendenze temporali di lungo periodo mantenendo una struttura totalmente convoluzionale. Nella formulazione proposta in [15], i blocchi combinano convoluzioni *pointwise* (1×1) e *depthwise* per ampliare progressivamente il *receptive field* con un incremento contenuto del costo computazionale.

Nelle applicazioni *embedded*, come gli hearing aids, questo aspetto è determinante, perché oltre alla qualità di denoising il modello deve rispettare anche i vincoli stringenti di cui già ampiamente discussi in precedenza. Nella proposta originale, la TCNN è inserita in una struttura encoder–decoder convoluzionale 2D che ricostruisce direttamente il segnale temporale, ma questa scelta tende ad aumentare la complessità totale del sistema.

Per ovviare a tale problema è stata introdotta la STFT-TCNN [16], in cui il segnale viene trasformato nel dominio tempo–frequenza tramite la STFT descritta nel Capitolo 4, così che la rete operi direttamente sulla magnitudine spettrale. La rete non produce quindi la *waveform* in uscita, ma stima una **maschera** applicata allo spettro complesso del segnale rumoroso; da questo, tramite ISTFT, si ricostruisce il segnale nel dominio del tempo.

5.1 Pipeline di elaborazione

Come già discusso in precedenza, il segnale osservato segue il modello additivo riportato in Eq. (3.1). La pipeline che il segnale audio deve attraversare è schematizzata in Figura 5.1 e può essere letta in cinque passaggi.

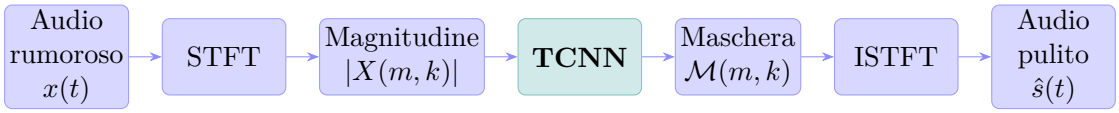


Figura 5.1: Pipeline di elaborazione della STFT-TCNN.

1. **Trasformazione del dominio.** Il segnale rumoroso viene convertito tramite la STFT definita in Eq. (4.1) nello spettro complesso

$$X(m,k) = \mathrm{STFT}\{x(t)\} \tag{5.1}$$

   dove m è l'indice del frame temporale e k quello del bin di frequenza.

2. **Estrazione della magnitudine.** Dallo spettro complesso si estrae la sola magnitudine

$$|X(m,k)| = \sqrt{\Re\{X(m,k)\}^2 + \Im\{X(m,k)\}^2} \tag{5.2}$$

   che costituisce l'input della rete. Con la configurazione base, l'asse in frequenza ha 257 bin, quindi il tensore di input ha forma $[\mathrm{batch}, 257, T]$, dove T è il numero di frame temporali.

3. **Elaborazione TCNN.** La magnitudine attraversa una sequenza di blocchi convoluzionali che modellano il contesto temporale. La struttura interna è descritta nella Sezione 5.2.

4. **Stima della maschera.** La rete $\mathcal{F}_\theta$, di parametri θ, stima una maschera reale

$$\mathcal{M}(m,k) = \mathcal{F}_\theta\Big(|X(m,k)|\Big) \tag{5.3}$$

   che viene applicata *elementwise* allo spettro complesso rumoroso:

$$\hat{S}(m,k) = \mathcal{M}(m,k) \cdot X(m,k) \tag{5.4}$$

5. **Ritorno al dominio di partenza.** Lo spettro stimato viene riconvertito con ISTFT:

$$\hat{s}(t) = \mathrm{ISTFT}\{\hat{S}(m,k)\} \tag{5.5}$$

In questa pipeline la rete agisce di fatto sulla sola ampiezza, dato che la maschera $\mathcal{M}(m,k)$, essendo reale, la scala a ciascun bin senza modificarne la fase, dove quest'ultima viene riutilizzata dal segnale rumoroso solo durante la fase di ricostruzione. Nel corso del training, la funzione obiettivo è definita come MSE tra lo spettro complesso stimato $\hat{S}(m,k)$ e quello di riferimento $S(m,k)$, calcolato

separatamente sulla parte reale e su quella immaginaria e mediato sugli F bin di frequenza e su T frame:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{FT} \sum_{m,k} \left(\left| \Re\{\hat{S}(m,k)\} - \Re\{S(m,k)\} \right|^2 + \left| \Im\{\hat{S}(m,k)\} - \Im\{S(m,k)\} \right|^2 \right) \tag{5.6}$$

5.2 Struttura della rete

L'architettura è costruita per livelli. Il punto di partenza è il blocco residuo, ossia l'unità elementare di elaborazione, dalla cui ripetizione in cascata nasce la rete completa.

5.2.1 Il blocco residuo

Il *ResBlock* rappresenta il core fondamentale, qui avviene l'apprendimento vero e proprio delle caratteristiche della voce. La sua struttura interna, illustrata in Figura 5.2, è una sequenza di operazioni convoluzionali, di normalizzazioni e di attivazioni, racchiuse da una connessione residua che somma l'ingresso all'uscita.

Prima di descrivere le singole operazioni conviene richiamare che cosa fa una convoluzione, perché è da questa che dipendono sia la qualità sia il costo della rete. Un livello convoluzionale fa scorrere un piccolo insieme di pesi, il *kernel*, lungo il segnale di ingresso: in ogni posizione il filtro calcola una somma pesata dei campioni che ricopre e ne produce uno in output. Ripetendo l'operazione lungo tutta la sequenza si ottiene il segnale filtrato.

Si distinguono due misure fondamentali: il numero di **parametri allenabili** che determina l'occupazione di memoria (FLASH e RAM) e il numero di **operazioni in virgola mobile** (*FLOP*) che determina il tempo di inferenza e cresce con l'aumento dei frame T elaborati. In una convoluzione 1D ordinaria ogni canale di uscita combina tutti i canali di ingresso su una finestra di K frame, quindi servono $C_{in} \cdot C_{out} \cdot K$ pesi e ripetendo l'operazione su T frame, il costo di calcolo è di circa $2\,C_{in}\,C_{out}\,K\,T$ FLOP, dove il fattore 2 conta la moltiplicazione e la somma di ogni *tap* del filtro. Entrambe le quantità crescono rapidamente con l'aggiunta di canali, ed è proprio per contenerle che il blocco residuo spezza la convoluzione ampia in tre passaggi più economici, ciascuno con un compito ben definito. La Tabella 5.1 ne riassume il costo, mentre le Figure 5.3, 5.4 e 5.5 ne illustrano graficamente la forma dell'ingresso, del kernel e dell'uscita.

- **Convoluzione *pointwise* (1×1).** Il kernel è largo un solo frame lungo l'asse temporale, non guarda i frame vicini e quindi non aggrega informazione nel

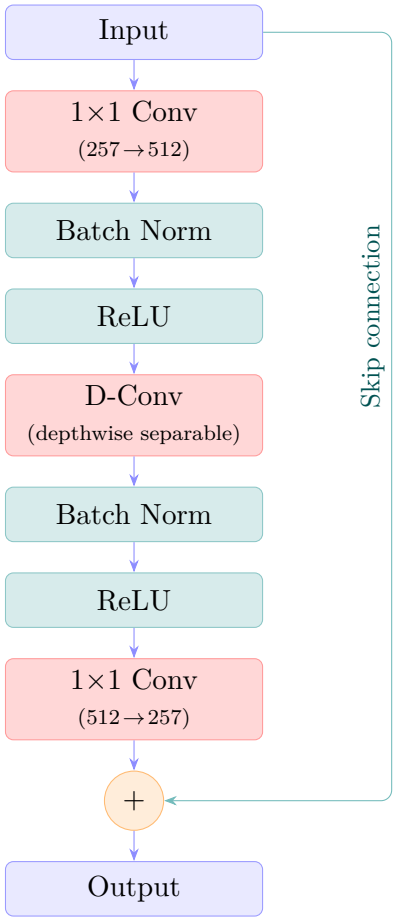


Figura 5.2: Struttura interna del blocco residuo (ResBlock): la convoluzione pointwise iniziale espande i canali, la D-Conv dilatata elabora il contesto temporale, e la pointwise finale li riporta alla dimensione di ingresso; la connessione residua somma l'ingresso originale all'uscita del ramo convoluzionale.

tempo. Si limita a ricombinare tra loro i canali di uno stesso frame come una matrice di pesi applicata in modo identico a ogni istante. Qui il suo compito è proiettare i 257 canali di ingresso in uno spazio più ampio, così da dare alla rete più capacità rappresentativa prima dell'elaborazione vera e propria. Poiché il kernel copre un solo frame, il conteggio dei pesi si riduce al prodotto $C_{in} \cdot C_{out}$, senza il fattore K di una convoluzione estesa nel tempo, e il costo di calcolo scende a circa $2\,C_{in}\,C_{out}\,T$ FLOP.

- **Convoluzione *depthwise-separable dilatata* (D-Conv).** È il passaggio in cui la rete aggrega effettivamente l'informazione lungo l'asse temporale, e lo fa fattorizzando l'operazione in due fasi più economiche. Nella prima, *depthwise*,

a ciascun canale è associato un filtro indipendente che opera unicamente su di esso: si estrae così la struttura temporale senza mescolare canali distinti, riducendo il numero di pesi a $C_{in} \cdot K$. Nella seconda, *pointwise* (1×1), i canali vengono ricombinati tra loro e la dimensionalità è riportata al valore di partenza, recuperando la capacità di correlazione tra canali che la sola depthwise non possiede. Il costo complessivo, pari a $C_{in}\,K + C_{in}\,C_{out}$ pesi, resta una frazione di quello di una convoluzione ordinaria a parità di contesto osservato.

A completare l'operazione interviene la **dilatazione**: il filtro depthwise non legge frame contigui, ma frame distanziati di un fattore d (*dilation*), ampliando l'intervallo temporale abbracciato senza introdurre pesi aggiuntivi. Il contributo di questo meccanismo al campo ricettivo della rete è quantificato nella Sezione 5.3.

- **Normalizzazione e attivazione.** Tra una convoluzione e l'altra il blocco inserisce due operazioni che non alterano il contenuto spettrale, ma rendono l'addestramento più stabile ed efficace. La *Batch Normalization* riporta le attivazioni di ciascun canale a una scala comune, infatti, evita che i valori crescano o si appiattiscano strato dopo strato e permette di usare learning rate più alti, accelerando la convergenza. La *ReLU* è invece la non linearità vera e propria, che azzera i valori negativi lasciando passare quelli positivi. Senza una funzione di questo tipo l'intera catena di convoluzioni collasserebbe in un'unica trasformazione lineare, incapace di separare il rumore dalla voce.

Operazione	Parametri	FLOP ($\approx$)
Conv 1D ordinaria	$C_{in}\,C_{out}\,K$	$2\,C_{in}\,C_{out}\,K\,T$
Pointwise (1×1)	$C_{in}\,C_{out}$	$2\,C_{in}\,C_{out}\,T$
Depthwise (kernel K)	$C_{in}\,K$	$2\,C_{in}\,K\,T$

Tabella 5.1: Costo delle convoluzioni impiegate nel ResBlock

L'intero ramo è racchiuso da una **connessione residua**: l'uscita del ramo convoluzionale viene sommata all'ingresso del blocco. Questo meccanismo, ereditato dalle architetture residue, facilita la propagazione del gradiente attraverso reti profonde e permette a ciascun blocco di apprendere una correzione rispetto alla rappresentazione in ingresso, anziché una trasformazione completa.

5.2.2 La rete completa

La rete completa si ottiene impilando i blocchi residui su due livelli gerarchici. Più *ResBlock* consecutivi, con dilatazione crescente, formano un **TCNBlock**;

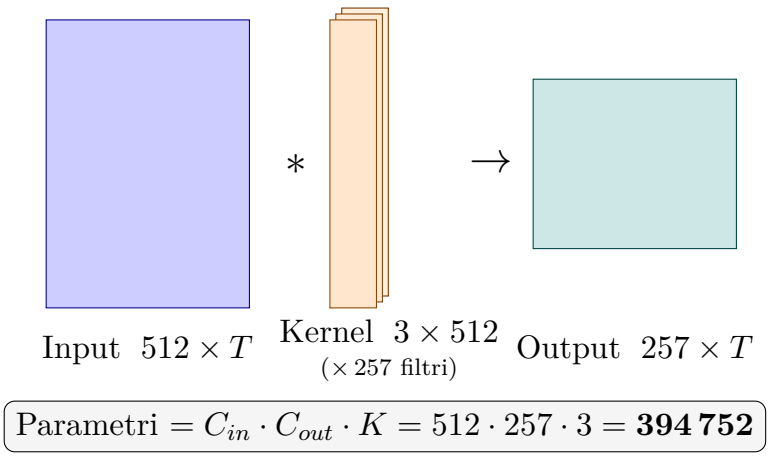


Figura 5.3: Convoluzione 1D ordinaria. Un kernel che copre tutti i 512 canali di ingresso per produrre ciascuno dei 257 canali di uscita, per un totale di 394.752 parametri.

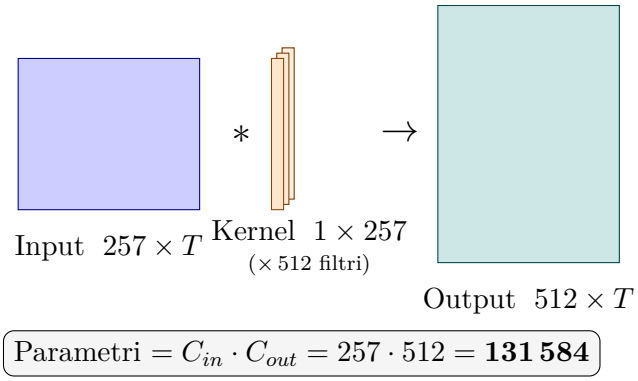


Figura 5.4: Convoluzione *pointwise* 1×1: il kernel copre un solo frame e si limita a ricombinare i canali, impiegando 131.584 parametri.

più *TCNBlock* in cascata formano la rete. La Figura 5.6 riporta lo schema della configurazione di riferimento, composta da $N = 2$ TCNBlock, ciascuno contenente $M = 3$ ResBlock.

Questa organizzazione consente di aggregare progressivamente informazioni temporali su finestre sempre più ampie del segnale, mantenendo al contempo un numero contenuto di parametri e una complessità computazionale compatibile con l'esecuzione su dispositivi embedded. La struttura non è rigida, ma parametrizzata, cioè agendo sugli iperparametri se ne modifica il comportamento, in particolare la quantità di contesto temporale (memoria latente) e il costo computazionale. I principali parametri configurabili sono:

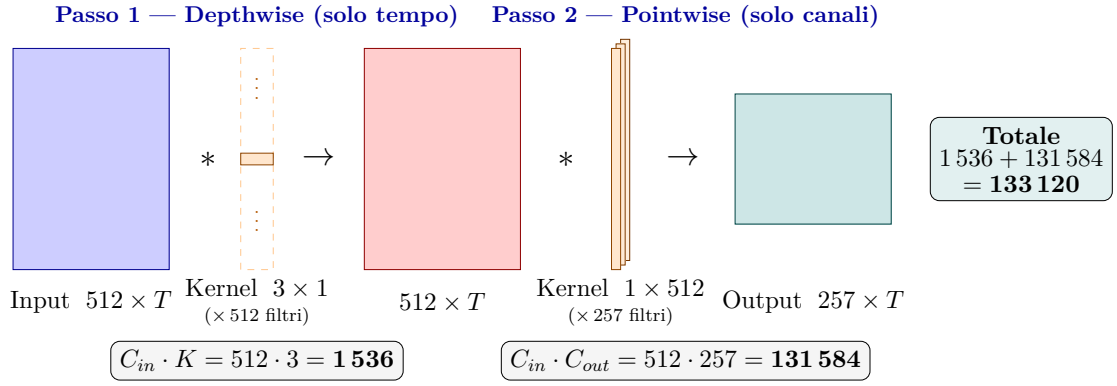


Figura 5.5: Convoluzione *depthwise-separable*. Il passo depthwise elabora ogni canale indipendentemente lungo il tempo, quello pointwise successivo ricombina i canali, per 133.120 parametri complessivi — una frazione dei 394.752 della convoluzione ordinaria equivalente.

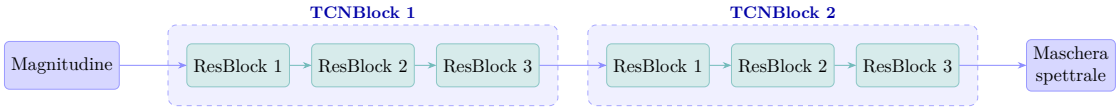


Figura 5.6: Schema dell'architettura della TCNN utilizzata nel modello STFT-TCNN.

- `n_blocks` (N) — numero di TCNBlock in cascata;
- `num_layers` (M) — numero di ResBlock per ogni TCNBlock;
- `kernel_size` (K) — ampiezza del kernel delle convoluzioni dilatate;
- `init_dilation` (D) — base della progressione della dilatazione;
- `tcn_latent_dim` — dimensione dello spazio latente delle convoluzioni *pointwise*.

Aumentare N o M significa estendere il *receptive field* e la capacità del modello, ma anche incrementare il costo per frame, infatti, la scelta di questi valori deriva da un compromesso diretto tra qualità del denoising e rispetto dei vincoli. La configurazione di riferimento adottata da STMicroelectronics fissa $N = 2$, $M = 3$, $K = 3$, $D = 2$ e `tcn_latent_dim`= 512.

5.3 Configurazione di riferimento

L'architettura originariamente proposta da STMicroelectronics è concepita per un utilizzo **offline**, mentre nel caso di studio affrontato in questo lavoro il sistema deve operare in modalità **streaming**. Nella sua formulazione di base la rete impiega convoluzioni non causali, con un *padding* simmetrico che le consente di sfruttare sia il contesto passato sia quello futuro: una scelta che migliora la qualità della ricostruzione, ma fa dipendere la stima del frame corrente da campioni non ancora disponibili in tempo reale. È questa dipendenza dal contesto futuro a determinare la latenza algoritmica del modello, che verrà quantificata nella Sezione 5.3.1 a partire dal receptive field.

La dimensionalità dell'input della rete è strettamente vincolata ai parametri della STFT introdotti nel Capitolo 4. Fissando il numero di punti della trasformata a $N_{FFT} = 512$, il numero di canali di ingresso e di uscita del modello risulta essere $\frac{N_{FFT}}{2} + 1 = 257$. Per riprodurre fedelmente il baseline, in questo capitolo si adottano i parametri STFT di riferimento di STMicroelectronics: segnali campionati a 16 kHz, finestra di analisi di 400 campioni (25 ms) e passo di avanzamento di 160 campioni (10 ms).

5.3.1 Il campo ricettivo

Il *receptive field* di una rete convoluzionale è il numero di frame temporali in ingresso che concorrono a determinare un singolo frame in uscita. Esso, in pratica, quantifica l'ampiezza della finestra di contesto che la rete è in grado di osservare.

Una convoluzione con kernel di ampiezza K e dilatazione d legge esattamente K frame distanziati di d posizioni l'uno dall'altro. Di conseguenza il filtro copre un intervallo di $(K-1)\,d+1$ frame ed estende il campo ricettivo di $(K-1)\,d$ frame rispetto allo strato precedente. Impostando $d_i = D^{\,i-1}$ sugli strati successivi ($i = 1, 2, \dots$), la dilatazione permette quindi di allargare il contesto in modo esponenziale senza aumentare il numero di pesi del kernel.

Quando più convoluzioni sono impilate, i rispettivi contributi $(K-1)\,d_i$ si sommano. Partendo da un singolo frame (il termine 1) e accumulando il contributo di ogni strato, l'ampiezza complessiva del *receptive field* R è data da:

$$R = 1 + \sum_i (K-1)\,d_i \tag{5.7}$$

Nella configurazione di riferimento ogni convoluzione dilatata usa kernel $K = 3$ e *stride* unitario, con dilatazione che cresce esponenzialmente secondo $d_i = 2^{\,i-1}$, dove $i = 1, \dots, M$ indicizza i ResBlock all'interno del TCNBlock. I tre ResBlock di ciascun TCNBlock assumono quindi dilatazioni $\{1, 2, 4\}$, e con $N = 2$ la rete

contiene sei convoluzioni dilatate con dilatazioni $\{1, 2, 4, 1, 2, 4\}$. La Tabella 5.2 riporta il contributo di ciascuno strato.

Strato i	TCNBlock	Dilatazione d_i	Contributo $(K-1)\,d_i$	R parziale
1	1	1	2	3
2	1	2	4	7
3	1	4	8	15
4	2	1	2	17
5	2	2	4	21
6	2	4	8	29

Tabella 5.2: Contributo di ciascuna convoluzione dilatata al *receptive field* complessivo.

Applicando l'Eq. (5.7), la somma dei sei contributi vale $2+4+8+2+4+8 = 28$, da cui:

$$R = 1 + 28 = 29 \text{ frame temporali.}$$

Con un passo STFT di 10 ms, i 29 frame del campo ricettivo coprono un intervallo di circa $(29-1) \times 10 = 280$ ms di segnale. Trattandosi di una rete non causale a *padding* simmetrico, questo contesto è ripartito in modo bilanciato tra passato e futuro: la stima del frame corrente richiede dunque circa 140 ms di campioni futuri (*look-ahead*).

5.3.2 Tabella di riferimento

I valori prestazionali e di occupazione di risorse del modello STFT-TCNN, pubblicati da STMicroelectronics nello *stm32ai-modelzoo* [16], costituiscono i **valori di riferimento** (*ground truth*) adottati in questo lavoro come termine di paragone. I risultati sperimentali presentati nei capitoli successivi sono confrontati rispetto a questo baseline. La Tabella 5.3 ne riassume le metriche, per il modello in formato *float32* e nella versione quantizzata a 8 bit (QDQ int8), valutato sul dataset Valentini.

Modello	Formato	RAM	FLASH	Parametri	PESQ	STOI	SNR	SI-SNR	MSE
STFT-TCNN	float32	7.05 KB	6.13 MB	1.594.368	2.480	0.932	18.190	18.104	1.136e-4
STFT-TCNN	QDQ int8	101.79 KB	1.80 MB	1.594.368	2.372	0.932	18.190	18.100	1.109e-4

Tabella 5.3: Tabella di riferimento del modello baseline STFT-TCNN [16], nei formati *float32* e quantizzato *int8*, valutato sul dataset Valentini (VoiceBank-DEMAND).

Capitolo 6

Loud-loss

Il modello STFT-TCNN descritto nel capitolo precedente è addestrato minimizzando l'errore quadratico medio (MSE) sullo spettro complesso ricostruito. Sebbene tale funzione sia ampiamente utilizzata per la task sviluppata in questa tesi, essa presenta un limite di fondo, dato che il calcolo dell'errore è puramente matematico e non riflette la reale qualità della percezione uditiva umana, infatti, penalizza in modo uniforme gli errori su tutto lo spettro e spinge così i modelli neurali a concentrarsi sulle componenti a bassa frequenza, che possiedono intrinsecamente un'energia più elevata, trascurando però tutti quei dati che, pur avendo minore energia, sono percettivamente decisive per l'intelligibilità del parlato.

Per superare questa discrepanza, prendendo ispirazione dal lavoro di Li et al. [17], è stata introdotta la *Loud-loss*, ossia una funzione di perdita percettivamente ponderata e fondata su principi psicoacustici. Sfruttando le curve di isofonia, quest'ultima assegna all'errore di ricostruzione pesi dipendenti dalla frequenza, in modo che le deviazioni vengano penalizzate proporzionalmente alla sensibilità uditiva umana. Trattandosi inoltre di una *loss* indipendente dall'architettura, può essere applicata al modello senza apportare alcuna modifica strutturale.

6.1 Il modello psicoacustico e la scala Phon

Per comprendere l'assegnazione dei pesi è necessario introdurre il concetto di percezione soggettiva del suono, quantificata attraverso la scala *Phon*. L'orecchio umano non percepisce l'intensità acustica (SPL) in modo lineare su tutte le frequenze: la banda udibile si estende indicativamente tra 20 Hz e 20 kHz, ma la sensibilità non è uniforme e raggiunge il massimo attorno a 3–4 kHz [18]. Una conseguenza diretta di questa non uniformità è che per essere percepiti allo stesso volume due toni di frequenza diversa devono possedere un SPL fisico differente: dove l'orecchio è poco

sensibile serve una pressione sonora maggiore, dove è molto sensibile ne basta una ridotta.

Le curve di isofonia danno una rappresentazione quantitativa di questa relazione: ogni curva unisce le coppie di frequenza e SPL fisico che un ascoltatore percepisce con il medesimo volume. A essere uguale è solo l'intensità percepita, non il suono nel suo complesso. L'unità di misura di questo livello percepito è il *Phon*: per definizione, il valore in *Phon* di una curva coincide con l'SPL in decibel del tono di riferimento a 1000 Hz che le appartiene. La formulazione proposta da Li et al. adotta la curva dei 40 phon, riportata in Figura 6.1, poiché questo livello approssima il volume medio del tipico parlato conversazionale e fornisce quindi i pesi più adatti a ottimizzare l'udibilità della voce.

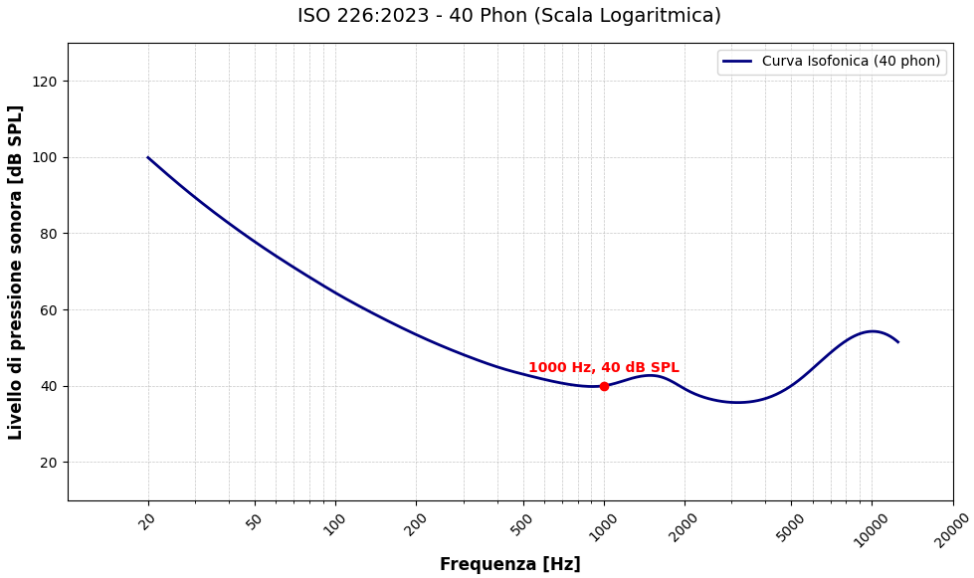


Figura 6.1: Curva isofonica a 40 phon (ISO 226:2023).

La curva riflette questa proprietà: nella zona di massima sensibilità, attorno a 3–4 kHz, è sufficiente un SPL ridotto (poco oltre i 35 dB) per raggiungere i 40 phon, mentre allontanandosi da questa regione ne occorre uno maggiore. Da questa relazione la Loud-loss deriva i propri pesi, come formalizzato nella sezione seguente.

6.2 Formulazione matematica

La Loud-loss opera sul confronto tra lo spettro di magnitudine stimato dal modello $|\hat{S}(m,k)|$ e quello di riferimento $|S(m,k)|$ secondo la notazione introdotta nel Capitolo 5. Il calcolo si articola in quattro fasi sequenziali:

1. **Conversione.** Le due magnitudini vengono convertite nel dominio logaritmico della potenza, espresso in decibel, per allinearlo alla percezione umana dell'intensità. Un piccolo termine ϵ evita instabilità numeriche in corrispondenza dei valori nulli:

$$P = 10\ \log_{10}\Big(|S|^2 + \epsilon\Big), \qquad \hat{P} = 10\ \log_{10}\Big(|\hat{S}|^2 + \epsilon\Big) \tag{6.1}$$

2. **Suddivisione in sottobande.** Lo spettro prodotto dalla STFT è una griglia uniforme di *bin*, ed ognuno copre lo stesso intervallo in Hz. L'orecchio discrimina bene le componenti gravi, mentre tende a fondere quelle acute in bande percettive sempre più larghe. Per pesare l'errore coerentemente con questa proprietà, i due spettri di potenza precedentemente calcolati vengono raggruppati in $K = 25$ sottobande spaziate secondo la scala Mel, che modella la risoluzione non uniforme del sistema uditivo. La conversione tra frequenza in Hz e scala Mel è data da

$$m(f) = 2595\ \log_{10}\Big(1 + \frac{f}{700}\Big), \qquad f(m) = 700\ \Big(10^{\frac{m}{2595}} - 1\Big) \tag{6.2}$$

   Per costruire le bande si individuano prima $K + 2$ centri equispaziati nell'intervallo tra 0 e $\frac{f_s}{2}$. Riportando questi centri in Hz, la natura logaritmica della scala fa sì che punti uniformi in Mel diventino fitti alle basse frequenze e via via più radi alle alte: si ottengono quindi molte bande strette dove l'udito è più selettivo e poche bande larghe dove lo è meno.

   Inoltre, quest'ultime risultano non essere disgiunte ma parzialmente sovrapposte, infatti, si può notare come l'i-esima sottobanda si estende dal centro i al centro $i+2$ dove la metà superiore di ciascuna di esse coincide così con la metà inferiore di quella seguente, e ogni bin centrale ricade in due bande adiacenti, con una sovrapposizione del 50% (Figura 6.2). In questo modo si evitano confini netti tra bande contigue e i pesi percettivi variano in modo graduale lungo lo spettro. Si indicano con P^i e $\hat{P}^i$ le porzioni di spettro relative alla banda i, con $i \in [0, K-1]$.

3. **Calcolo dell'errore per sottobanda.** Per ciascuna delle K sottobande si calcola l'MSE indipendente, mediato sui F_i bin della banda e sui T frame temporali:

$$\mathcal{L}^i_{sub} = \frac{1}{F_i\, T} \sum_{f,\,t} \Big(P^i(f,t) - \hat{P}^i(f,t)\Big)^2 \tag{6.3}$$

4. **Funzione di perdita.** Infine, ogni errore di sottobanda viene moltiplicato per un peso scalare w^i derivato dalla curva di isofonia. Il peso è il rapporto tra

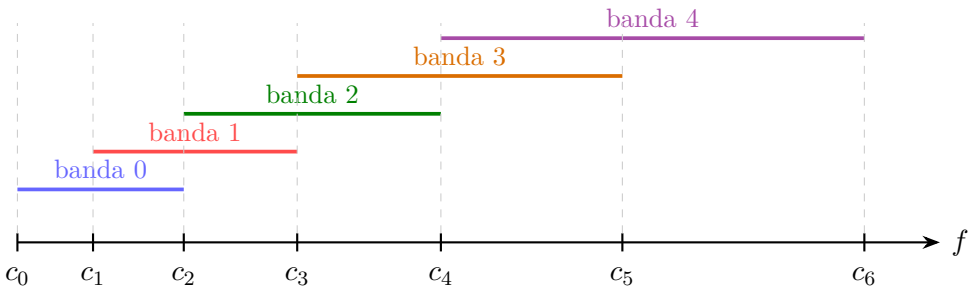


Figura 6.2: Costruzione delle sottobande sull'asse delle frequenze. I centri c_i sono equispaziati in scala Mel.

l'SPL di riferimento a 1000 Hz (SPL_{ref}) e l'SPL corrispondente alla frequenza centrale della banda (SPL_i):

$$\mathcal{L}_{loud} = \sum_{i=0}^{K-1} w^i \, \mathcal{L}_{sub}^i \qquad \text{con} \qquad w^i = \frac{SPL_{ref}}{SPL_i} \tag{6.4}$$

Il meccanismo di pesatura realizza così l'obiettivo percettivo. Nelle bande in cui l'udito è più sensibile l'SPL richiesto sulla curva dei 40 phon è basso, quindi $w^i > 1$ e l'errore viene amplificato, al contrario, dove l'udito è poco sensibile l'SPL è elevato, quindi $w^i < 1$ e l'errore viene attenuato. La rete è così spinta a ricostruire con maggiore fedeltà le componenti spettrali percettivamente più rilevanti per l'intelligibilità del parlato.

6.3 Variante con SI-SNR

Sebbene $\mathcal{L}_{loud}$ migliori in modo efficace la qualità percettiva, da sola non impone un vincolo esplicito sulla coerenza temporale della forma d'onda ricostruita. Per questo motivo è stata introdotta una variante che aggiunge una componente nel dominio del tempo basata sul SI-SNR (Sezione 3.3) con l'obiettivo di preservare meglio la struttura temporale del segnale e ridurre gli artefatti residui.

La funzione di perdita finale è quindi definita come somma pesata dei due contributi:

$$\mathcal{L}_{totale} = \alpha \, \mathcal{L}_{loud} + \beta \, \mathcal{L}_{SI\text{-}SNR} \tag{6.5}$$

dove α e β controllano, rispettivamente, il bilanciamento tra qualità percettiva e fedeltà temporale. L'uso di due pesi distinti è necessario perché i termini sono calcolati in domini differenti e presentano ordini di grandezza diversi. Nella configurazione adottata in questo lavoro si è posto $\alpha = 0.01$ e $\beta = 1.0$.

6.4 Risultati sperimentali

6.4.1 Setup

I test sono stati eseguiti in addestramento usando una GPU NVIDIA A100, mantenendo una configurazione coerente con il setup di riferimento STMicroelectronics adottato per il baseline e discusso nel Capitolo 5. La configurazione completa è riassunta di seguito:

- **Modello.** STFT-TCNN nella stessa configurazione architetturale originale.
- **STFT.** Parametri secondo il significato discusso nel Capitolo 4: `n_fft`= 512, `hop_length`= 160, `win_length`= 400, finestra di *Hann*, `center`=`true`, frequenza di campionamento 16 kHz.
- **Dataset.** VoiceBank-DEMAND [19] (noto anche come dataset *Valentini*, dal nome della curatrice Valentini-Botinhao), 56 speaker, segnali a 16 kHz, con validazione su 100 esempi selezionati dal training set.
- **Training.** Ottimizzatore Adam con *learning rate* iniziale 10^{-3}, *batch size* 16, fino a 200 epoche con *early stopping* sulla metrica di riferimento PESQ. Il *learning rate* è regolato da uno scheduler `ReduceLROnPlateau` che lo dimezza quando il PESQ di validazione non migliora per 3 epoche consecutive, fino a un valore minimo di 10^{-5}. I pesi delle due componenti della *loss* sono quelli definiti nella Sezione 6.3 (α, β).

6.4.2 Raccolta dei dati

La Tabella 6.1 riporta le metriche oggettive sul formato *float32*, confrontando il baseline STFT-TCNN che adotta la MSE rispetto alle due varianti.

Funzione di perdita	PESQ	STOI	SNR	SI-SNR	MSE
MSE [16]	2.480	0.932	18.190	18.104	1.136e-4
$\mathcal{L}_{loud}$	**2.860**	0.945	17.518	17.425	1.178e-4
$\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$	2.838	**0.947**	**18.315**	**18.261**	**1.077e-4**

Tabella 6.1: Metriche oggettive (formato float32) sul dataset Valentini.

La Tabella 6.2 riporta le medesime metriche per i modelli quantizzati a 8 bit (QDQ int8), il formato effettivamente rilevante per l'esecuzione su dispositivo *embedded*. La quantizzazione restringe il vantaggio percettivo osservato in *float32*: le due varianti conservano il PESQ e lo STOI migliori, ma sulle metriche energetiche (SNR, SI-SNR e MSE) il baseline torna in vantaggio, segno che la versione *int8* ne preserva meglio la fedeltà di forma d'onda.

Funzione di perdita	PESQ	STOI	SNR	SI-SNR	MSE
MSE [16]	2.372	0.932	**18.190**	**18.100**	**1.109e-4**
$\mathcal{L}_{loud}$	2.473	0.941	17.439	17.335	1.230e-4
$\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$	**2.484**	**0.942**	17.843	17.762	1.231e-4

Tabella 6.2: Metriche oggettive (formato quantizzato QDQ int8) sul dataset Valentini.

Si è inoltre condotta una prova qualitativa su due clip rappresentativi della durata di circa 45 secondi ciascuno, in cui un interlocutore parla in presenza di un rumore di fondo non stazionario: nel primo il disturbo è rappresentato dal traffico stradale, nel secondo il brusio di una folla di persone. Ogni clip è stata elaborata dal modello nelle sue tre versioni dettate dalle funzioni di perdita e l'audio risultante è stato valutato con il framework DNSMOS, già descritto nel Capitolo 3: trattandosi di una metrica non intrusiva, è particolarmente adatta a queste clip reali, prive di un riferimento pulito. La Tabella 6.3 ne riporta i risultati separati per scenario, sul formato *float32*.

Scenario	Modello	OVRL	SIG	BAK	MOS
Traffico	MSE (baseline) [16]	2.74	3.19	3.58	**3.25**
	$\mathcal{L}_{loud}$	2.52	2.99	3.46	2.84
	$\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$	**2.81**	**3.28**	**3.58**	2.97
Folla	MSE (baseline) [16]	2.74	3.18	3.62	**3.32**
	$\mathcal{L}_{loud}$	2.65	3.04	3.65	3.06
	$\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$	**2.85**	**3.28**	**3.68**	3.09

Tabella 6.3: Metriche soggettive DNSMOS (formato float32) stimate su due clip audio

Per verificare la correttezza dell'addestramento, in Figura 6.3 si riportano gli andamenti della *training loss* e delle principali metriche di validazione per le due varianti. In entrambi i casi la loss decresce in modo regolare fino a stabilizzarsi, mentre le restanti metriche raggiungono un *plateau* senza successivo degrado, rispettivamente, l'MSE di validazione non risale e il PESQ non peggiora nelle epoche finali. Questo andamento suggerisce l'assenza sia di *underfitting* sia di *overfitting*. L'*early stopping*, inoltre, seleziona per ciascuna variante il modello in corrispondenza del massimo della metrica di validazione, escludendo per costruzione le epoche potenzialmente sovra-adattate.

6.4.3 Valutazione dei risultati

La variante della STFT-TCNN che adotta la sola $\mathcal{L}_{loud}$ al posto della MSE conferma in pieno il comportamento riportato in [17]: l'ottimizzazione percettiva migliora

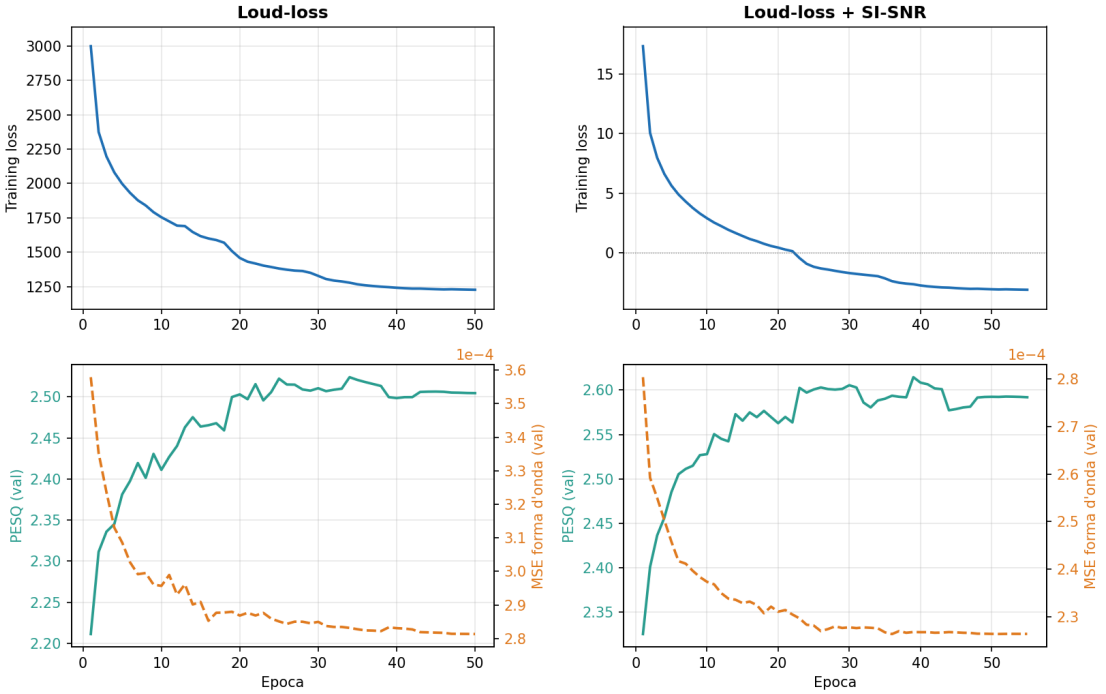


Figura 6.3: Andamento dell'addestramento per le varianti $\mathcal{L}_{loud}$ e $\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$.

in modo marcato la qualità percepita — il PESQ sale a 2.860 contro il 2.480 del baseline e lo STOI a 0.945 contro 0.932 — ma a scapito delle metriche energetiche nel dominio del tempo, con un calo dello SNR (da 18.190 a 17.518) e del SI-SNR (da 18.104 a 17.425). Si manifesta cioè il classico *trade-off* tra fedeltà percettiva e fedeltà di forma d'onda.

L'aggiunta del termine SI-SNR ribalta questo quadro, infatti, la variante $\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$ cede un margine minimo sul PESQ rispetto alla precedente, ma recupera la fedeltà temporale superando di fatto il baseline su tutte le metriche energetiche, con uno SNR di 18.315 (contro 18.190), un SI-SNR di 18.261 (contro 18.104) e l'MSE sulla forma d'onda più basso in assoluto (1.077e-4), mantenendo al contempo il miglior STOI (0.947).

Le metriche soggettive DNSMOS sui due scenari reali confermano la stessa gerarchia cioè che l'aggiunta del termine SI-SNR fa migliorare complessivamente i valori delle metriche in entrambi i casi, mentre la sola Loud-loss risulta la più debole. Il baseline conserva un leggero vantaggio unicamente sul MOS, indice che pondera diversamente la naturalezza complessiva del segnale.

Per queste ragioni la funzione di perdita adottata come riferimento per il resto del lavoro sarà proprio $\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$.

Capitolo 7

Modulo ERB

Il modello visto fin'ora tratta i 257 bin della STFT come canali indipendenti e di pari importanza. Questo capitolo introduce il **modulo ERB**, un blocco di compressione frequenziale, di ispirazione psicoacustica, posto a monte e a valle della rete con l'obiettivo di ridurre la dimensionalità delle feature (e quindi la complessità del modello) senza sacrificare la qualità percepita.

7.1 Equivalent Rectangular Bandwidth

Come anticipato nel Capitolo 4, la coclea scompone il suono nelle sue componenti di frequenza, comportandosi come una sorta di trasformata di Fourier biologica. Questa scomposizione, però, non ha risoluzione uniforme: ogni regione della coclea risponde a un piccolo intervallo di frequenze attorno a un valore centrale. Tale intervallo prende il nome di *banda critica* dove l'orecchio fonde assieme tutte le componenti che vi cadono dentro, senza distinguerle. Ciascuna di queste regioni si comporta come un singolo filtro passa-banda centrato sulla propria frequenza caratteristica f_c, poiché risponde al massimo ai toni vicini a f_c e sempre meno a quelli che se ne allontanano. La curva che ne descrive la risposta al variare della frequenza in ingresso prende il nome di *filtro uditivo* e ne rappresenta la forma, dato che un tono prossimo al centro eccita quella regione quasi per intero mentre uno lontano vi contribuisce appena. Misurate sperimentalmente, queste curve risultano irregolari e asimmetriche, perciò scomode da trattare nei calcoli.

Per aggirare questa difficoltà, la *Equivalent Rectangular Bandwidth* (ERB) ne propone una descrizione semplificata. A ciascun filtro uditivo si associa un filtro *rettangolare ideale*, centrato sulla stessa f_c e vincolato a rispettare due condizioni. Deve anzitutto avere la *stessa altezza* del filtro reale e deve inoltre lasciar passare la *stessa potenza* complessiva, racchiudendo quindi la stessa area. Fissati questi due vincoli la larghezza del rettangolo risulta univocamente determinata ed è proprio

questa a fornire la misura della banda critica (Figura 7.1). L'ERB condensa perciò in un unico numero l'ampiezza percettiva dell'udito attorno a quella frequenza.

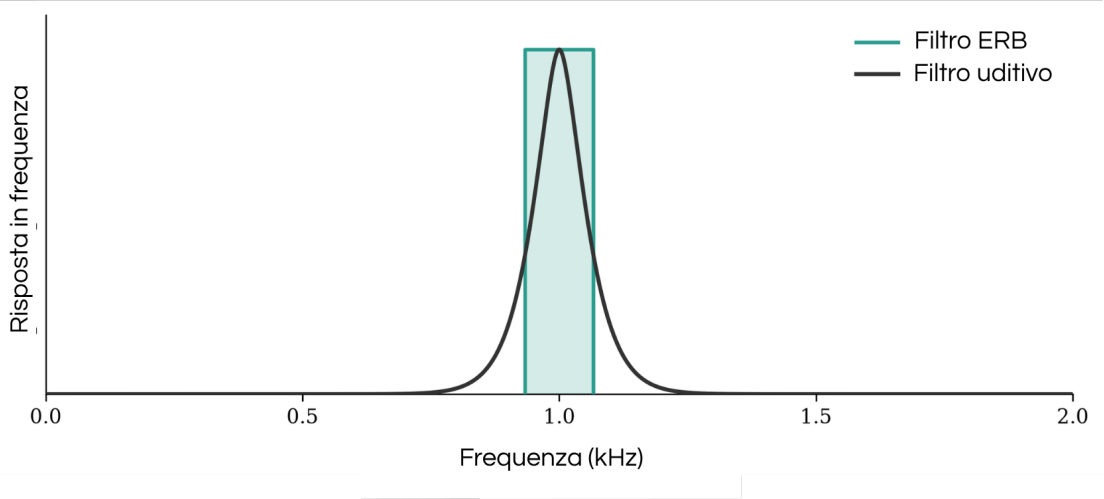


Figura 7.1: Approssimazione ERB del filtro uditivo: il rettangolo ha la stessa altezza di picco e la stessa area del filtro reale, e la sua larghezza definisce la banda critica.

La determinazione empirica dell'ERB in funzione della frequenza centrale f ha trovato la sua formulazione di riferimento nei lavori di Moore e Glasberg [20]:

$$ERB(f) = 24.7\left(4.37\,f + 1\right) \tag{7.1}$$

Questa relazione evidenzia come la risoluzione frequenziale dell'udito non sia uniforme, infatti, la larghezza di banda cresce con la frequenza centrale e, di conseguenza come già si è visto, l'orecchio risolve le basse frequenze con maggior dettaglio rispetto alle alte.

Tuttavia l'ERB non è l'unico modello che gode di questa proprietà. La letteratura propone altre scale psicoacustiche che descrivono la stessa risoluzione frequenziale decrescente, in particolare la scala *Mel* — già impiegata per la Loud-loss — e la scala *Bark*. Tutte producono bande strette alle basse frequenze e via via più larghe alle alte, e differiscono soprattutto per gli esperimenti psicoacustici da cui derivano. Per questo modulo di compressione la scelta è ricaduta sull'ERB, oggi considerata la stima più fedele della larghezza dei filtri uditivi, in particolare nella regione delle basse frequenze.

L'Eq. (7.1) descrive la *larghezza* di un singolo filtro, mentre ai fini della compressione conviene ragionare sulla corrispondente **scala ERB**. Si tratta di un nuovo asse delle frequenze, ottenuto deformando quello in Hz, lungo il quale la distanza fra due toni non si conta più in Hz bensì nel *numero di bande critiche* che li separano. Su questo asse spostamenti uguali corrispondono a differenze percettive uguali, perciò un piccolo intervallo in Hz copre molte ERB in basso e poche in alto. La si ricava integrando l'inverso della larghezza di banda dell'Eq. (7.1), che conduce alla forma chiusa

$$E(f) = 21.4 \log_{10}\Big(1 + 4.37\, f\Big), \qquad f \text{ in kHz} \tag{7.2}$$

e la relazione inversa, che dalla posizione sull'asse percettivo riporta alla frequenza in Hz,

$$f(E) = \Big(10^{\,E/21.4} - 1\Big)/4.37 \tag{7.3}$$

Nella costruzione del banco di filtri (Sezione 7.3) le due funzioni lavorano in coppia. La $E(f)$ trasforma una frequenza in Hz nella sua posizione sull'asse percettivo, dove le bande vengono disposte a passi regolari e perciò percettivamente equivalenti. La sua inversa $f(E)$ compie il cammino opposto e riporta quelle posizioni in Hz, così da individuare gli indici dei bin STFT su cui ogni banda va a poggiare.

7.2 Compressione delle feature e riduzione della complessità

La motivazione dietro l'inserimento di questo modulo all'interno della rete è duplice:

- **Motivazione percettiva.** Poiché la risoluzione uditiva decresce con l'aumentare della frequenza, rappresentare quelle alte con la stessa densità di bin di quelle basse è ridondante dal punto di vista percettivo. Raggruppando i bin ad alta frequenza in bande spaziate secondo questa scala, la dimensionalità delle feature si riduce senza perdere l'informazione uditivamente rilevante.
- **Motivazione computazionale.** Il costo delle convoluzioni *pointwise* presenti nei blocchi residui cresce con il prodotto $C_{in} \cdot C_{out}$ del numero di canali. Comprimere l'asse in frequenza da 257 a 129 canali, e mantenere la rete su tale dimensione, riduce quindi in modo sostanziale parametri e operazioni per frame.

Questo è il principio adottato da modelli a bassissima complessità come **PercepNet** [21], che opera su sole 32 bande triangolari, e **GTCRN** [22], che impiega un

banco di filtri ERB proprio per ridurre la ridondanza delle feature di ingresso, ed è da quest'ultimo che deriva la realizzazione adottata in questo lavoro.

7.3 Formulazione matematica

Il modulo opera sulla magnitudine spettrale $|X(m,k)|$ del segnale rumoroso in ingresso, frame per frame, e il procedimento si riassume in cinque passaggi:



Figura 7.2: Collocazione del modulo ERB nell'architettura ERB-TCNN.

1. **Ripartizione in banda bassa e banda alta.** I primi 65 bin, corrispondenti all'intervallo 0–2031 Hz, dove si concentra l'energia del parlato e la struttura del *pitch*, sono lasciati invariati preservando la piena risoluzione lineare. I restanti $257-65=192$ bin, corrispondenti a 2031–8000 Hz, vengono compressi in 64 bande percettive. Conviene adesso tradurre questo confine da numero di bin a frequenza: con `n_fft` $=512$ e $f_s=16$ kHz ogni bin copre $\frac{f_s}{\texttt{n_fft}}=31{,}25$ Hz e il bin di indice k cade a $\frac{k\cdot f_s}{\texttt{n_fft}}$. La banda alta parte dal bin 65, quindi il suo estremo inferiore è $f_{low}=\frac{65\cdot 16000}{512}=2031{,}25$ Hz cioè è la frequenza alla quale termina la regione a risoluzione lineare e comincia quella compressa.

2. **Posizionamento delle bande sulla scala.** Le bande non vengono distribuite uniformemente in Hz, ma in modo che risultino percettivamente equivalenti, per questo il posizionamento avviene sulla scala dell'Eq. (7.2), dove passi di uguale ampiezza corrispondono a bande di pari rilevanza uditiva. Si prende quindi l'intervallo $[E(f_{low}),\, E(8000\,\text{Hz})]$ e lo si divide in passi costanti, ottenendo $64+2=66$ punti. Ne servono due in più delle 64 bande perché ogni filtro triangolare poggia su tre punti consecutivi, le due basi e il vertice, come dettagliato nel punto seguente. Questi 66 valori vengono infine riportati in Hz tramite l'inversa $f(E)$ e arrotondati all'indice di bin STFT più vicino, dando i confini interi $b_0 < b_1 < \cdots < b_{65}$.

3. **Banco di filtri triangolari.** A partire da questi confini si costruisce una matrice di pesi $W \in \mathbb{R}^{64\times 192}$ in cui la i-esima banda è un filtro triangolare che sale linearmente da b_i a b_{i+1} e scende da b_{i+1} a b_{i+2}:

$$W_{i,j} = \begin{cases} \dfrac{j - b_i}{b_{i+1} - b_i} & b_i \le j < b_{i+1} \\ \dfrac{b_{i+2} - j}{b_{i+2} - b_{i+1}} & b_{i+1} \le j < b_{i+2} \\ 0 & \text{altrimenti} \end{cases} \tag{7.4}$$

La Figura 7.3 ne mostra l'andamento qualitativo.

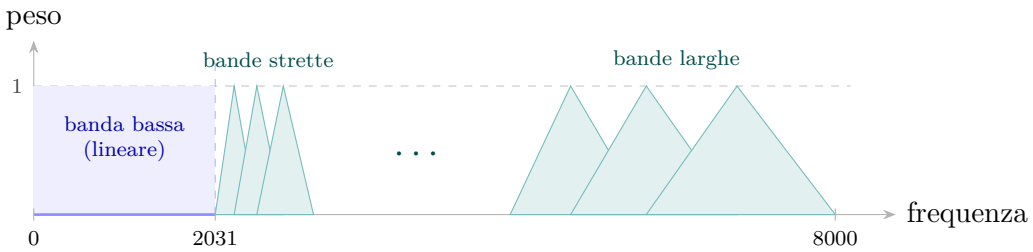


Figura 7.3: Schema qualitativo del banco di filtri del modulo ERB.

4. **Compressione.** La compressione b_m concatena la banda bassa della magnitudine $|X|$ invariata con la proiezione di quella alta sulle 64:

$$b_m(|X|) = \left[\, |X|_{0:65} \;;\; W\,|X|_{65:257} \,\right] \in \mathbb{R}^{129} \tag{7.5}$$

L'operazione $W\,|X|_{65:257}$ è un prodotto matriciale lungo l'asse della frequenza, implementato come convoluzione 1×1 sui canali. Il tensore risultante ha $65 + 64 = 129$ canali, sui quali opera il TCN descritto nel Capitolo 5.

5. **Espansione.** Dopo il TCN, la maschera nel dominio compresso $\mathcal{M}_c \in \mathbb{R}^{129}$ viene riportata ai 257 bin originari. La banda bassa è ricopiata invariata, mentre le 64 bande compresse sono riproiettate sui 192 bin alti tramite la *trasposta* del banco di filtri:

$$b_s(\mathcal{M}_c) = \left[\, \mathcal{M}_{c,\,0:65} \;;\; W^\top \mathcal{M}_{c,\,65:129} \,\right] \in \mathbb{R}^{257} \tag{7.6}$$

È importante sottolineare che $W^\top$ non è l'inversa di W perché la compressione dei valori è un operazione non invertibile, quindi l'espansione realizza una sintesi approssimata, in cui il valore di ciascuna banda viene ridistribuito sui bin del proprio supporto triangolare. Al risultato si applica infine l'attivazione finale (sigmoide o tangente iperbolica) che produce la maschera definitiva sui 257 bin.

Le matrici W e $W^\top$ sono **congelate** (`requires_grad = False`) cioè sono determinate analiticamente dalle Eq. (7.2)–(7.4) e non vengono modificate durante l'addestramento. Ciò le rende precomputabili come tabelle di *lookup* costanti.

7.4 Collocazione nell'architettura della rete

Il modulo ERB non sostituisce alcun elemento della pipeline del Capitolo 5, si inserisce immediatamente prima del primo blocco TCN effettuando una compressione e subito dopo l'ultimo, eseguendo una espansione, lasciando invariata la struttura interna dei blocchi residui. Il modello risultante, denominato **ERB-TCNN**, differisce dal baseline STFT-TCNN essenzialmente per la dimensione su cui lavora la rete, come riassunto nella Tabella 7.1.

Caratteristica	STFT-TCNN	ERB-TCNN
N# canali	257 (bin lineari)	129 (65 lineari + 64 ERB)
Dimensione latente	512	129
Risoluzione alle alte frequenze	lineare	compressa

Tabella 7.1: Confronto architetturale tra il baseline STFT-TCNN e ERB-TCNN.

A differenza dello STFT-TCNN, in cui i blocchi residui espandono i canali a una dimensione latente di 512, l'ERB-TCNN mantiene la rete sui 129 canali e la combinazione tra compressione in frequenza e assenza di espansione latente è quindi all'origine della riduzione del costo del modello.

7.5 Risultati sperimentali

L'addestramento di questa rete prevede l'uso delle due funzioni di perdita descritte ampiamente nel precedente capitolo, con la stessa procedura di training vista in Sezione 6.4.1. Diversamente dal capitolo precedente, tuttavia, `hop_length` e profondità della rete sono già impostati ai valori risultati migliori dalla sperimentazione del Capitolo 9 (128 campioni e 5 livelli per blocco, al posto del riferimento STMicroelectronics): il modulo ERB è stato infatti sviluppato in parallelo a quella ottimizzazione, qui anticipata per coerenza espositiva. La valutazione procede su due piani complementari: la *qualità* della voce ricostruita e il *costo* di esecuzione su dispositivo.

7.5.1 Qualità della ricostruzione

La Tabella 7.2 riporta le metriche oggettive, in formato *float32* e nella versione quantizzata *int8*, confrontando il baseline STFT-TCNN con le due varianti ERB-TCNN.

Formato	Modello	PESQ	STOI	SNR	SI-SNR	MSE
float32	STFT-TCNN (baseline) [16]	2.480	0.932	18.190	18.104	1.136e-4
	ERB-TCNN + $\mathcal{L}_{loud}$	2.751	0.941	17.160	17.101	1.383e-4
	ERB-TCNN + $\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$	**2.782**	**0.943**	**18.223**	**18.147**	**1.072e-4**
int8	STFT-TCNN (baseline) [16]	2.372	0.932	**18.190**	**18.100**	**1.109e-4**
	ERB-TCNN + $\mathcal{L}_{loud}$	2.551	0.940	16.944	16.856	1.419e-4
	ERB-TCNN + $\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$	**2.570**	**0.942**	18.005	17.917	1.118e-4

Tabella 7.2: Confronto delle metriche di training tra modello baseline e il modello che implementa il modulo ERB.

Il dato saliente è che il modulo ERB non degrada la qualità della voce, pur lavorando su molti meno canali. In *float32* la variante $\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$ supera il baseline su **tutte** le metriche mentre la sua versione quantizzata ridimensiona in parte questo vantaggio, com'era già emerso nel Capitolo 6: l'aggiunta del termine SI-SNR nella funzione di perdita ancora una volta conserva il PESQ migliore (2.570 contro 2.372) e lo STOI più alto ma sulle metriche energetiche (SNR, SI-SNR e MSE) il baseline torna in vantaggio. Si ritrova così la stessa gerarchia già vista in precedenza, dove la sola $\mathcal{L}_{loud}$ sacrificava gli indici di forma d'onda senza primeggiare su alcuna metrica, mentre $\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$ ricomponeva il *trade-off* confermandosi la variante migliore. La compressione ERB, quindi, riorganizza le feature senza intaccare la qualità della voce ricostruita e in *float32*, anzi, la migliora su tutti gli indici.

7.5.2 Costo su dispositivo

Misurata la qualità, resta da quantificare il risparmio di risorse. Il termine di paragone è la tabella di verità di STMicroelectronics (Tabella 5.3); l'ERB-TCNN è stato compilato e profilato in proprio con STM32CubeAI Studio sul target STM32N6. Entrambi i modelli sono considerati in formato *int8*, l'unico rilevante per l'esecuzione su dispositivo *embedded*. La Tabella 7.3 ne riassume occupazione di memoria e latenza.

Si dimostra che aver introdotto questo componente all'interno della rete riduce il numero di parametri da circa 1,59 milioni a 364 mila, un fattore di circa 4,4, l'occupazione di memoria FLASH scende da 1,80 MB a 531,5 KB (fattore $\approx 3{,}4$) e la RAM da 101,79 KB a 63,66 KB. Questi valori confermano per via sperimentale

Modello	Parametri	FLASH	RAM	Latenza
STFT-TCNN (baseline) [16]	1.594.368	1.80 MB	101.79 KB	51.01 ms
ERB-TCNN	363.846	531.5 KB	63.66 KB	15.91 ms

Tabella 7.3: Occupazione di risorse e latenza di inferenza dei due modelli in formato quantizzato *int8* sul target STM32N6.

la previsione della Sezione 7.2, ossia che la compressione dell'asse in frequenza unita all'assenza di espansione latente riduce in modo diretto il costo.

La conseguenza più importante riguarda la latenza. Meno parametri e meno operazioni significano un tempo di inferenza più breve: sullo stesso microcontrollore STM32N6, il tempo necessario a elaborare una finestra di 40 frame passa dai 51,01 ms del baseline ai 15,91 ms dell'ERB-TCNN, una riduzione di oltre tre volte. Trattandosi di reti **non causali**, addestrate e profilate su sequenze di blocchi di frame, questo valore è il costo dell'intera finestra e non di un singolo frame: il superamento di questo limite, con il passaggio a un'elaborazione causale *frame-by-frame*, è proprio l'oggetto del capitolo seguente. Questo guadagno non costa nulla in qualità, infatti questo modello risulta essere più leggero, più veloce e percettivamente migliore del baseline da cui parte.

Capitolo 8

Streaming real-time

Alla rete neurale attuale manca un requisito senza il quale l'intero sistema resta inutilizzabile nel contesto applicativo di questa tesi, ossia la capacità di operare in **streaming real-time**, elaborando il segnale *frame-by-frame* man mano che viene acquisito, con latenza minima e senza dipendere da campioni futuri. Questo capitolo affronta tale requisito, mostra perché la rete di partenza non lo soddisfaceva e presenta il **modello finale** effettivamente adottato.

8.1 Causalità e funzionamento real-time

Un sistema di elaborazione di sequenze si dice **causale** quando la sua uscita all'istante t dipende esclusivamente dagli ingressi presenti e passati, e mai da quelli futuri. Formalmente è detta $\hat{y}_t$ la stima prodotta dalla rete:

$$\hat{y}_t = \mathcal{F}\Big(x_0, x_1, \ldots, x_t\Big) \tag{8.1}$$

dove non compare alcuna dipendenza da $x_{t+1}, \ldots, x_T$. La causalità è condizione necessaria per l'elaborazione real-time di un flusso audio: se la stima del frame corrente richiedesse campioni futuri, il sistema dovrebbe attenderne l'arrivo prima di produrre un'uscita, introducendo una *latenza algoritmica* pari alla durata del contesto futuro richiesto. Tale latenza si somma a quella, inevitabile, della finestratura STFT discussa nel Capitolo 4.

Il modello da cui discende l'intera architettura — la TCNN di Pandey e Wang [15] — nasceva già adatto all'uso in tempo reale. Le TCNN sono infatti definite come reti convoluzionali per la modellazione di sequenze *con vincolo causale* [23]: il modulo temporale impiega convoluzioni dilatate che attingono unicamente al frame corrente e a quelli passati, garantendo, per usare le parole degli autori, che "non vi sia alcuna fuga di informazione dal futuro verso il passato". La dilatazione amplia il *receptive field* estendendolo all'indietro, mai in avanti.

8.2 La perdita di causalità nella baseline

La reimplementazione fornita da STMicroelectronics e adottata come baseline abbandona il vincolo causale. Nei blocchi residui, la convoluzione *depthwise* dilatata è configurata con *padding* simmetrico: l'uscita all'istante t attinge così sia ai frame passati sia a quelli *futuri* $(t + d_i)$, dove d_i è la dilatazione del layer. La rete diventa **non causale**: in cambio di un contesto bilaterale — che migliora la qualità della ricostruzione — introduce un look-ahead pari, per ciascun layer, alla sua dilatazione.

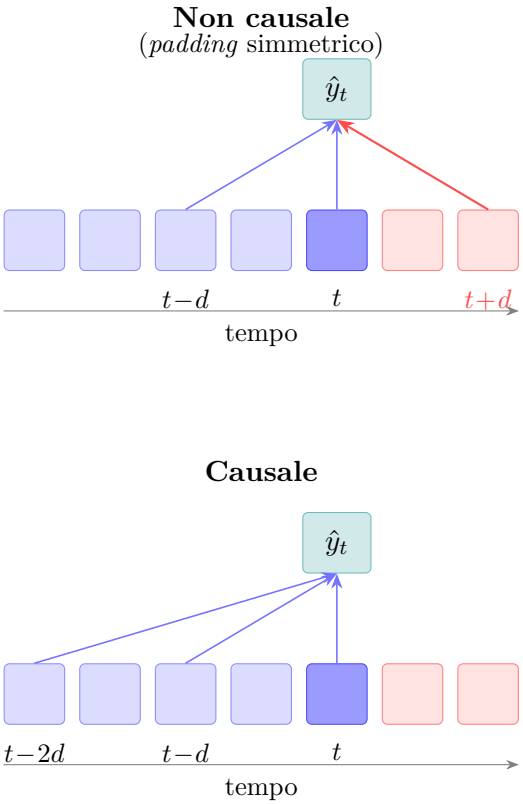


Figura 8.1: Confronto tra convoluzione causale e non causale su un singolo layer dilatato ($K = 3$, $d = 2$): nel pannello non causale (sopra) l'uscita dipende anche da un campione futuro $t + d$, mentre nel pannello causale (sotto) dipende solo da campioni presenti e passati.

Sommando i contributi di tutti i layer, la porzione di contesto futuro è $\sum_i d_i$, circa metà del campo ricettivo della rete e crescente con la sua profondità: per il baseline coincide con i ∼140 ms già quantificati nel Capitolo 5. Un ritardo simile è incompatibile con la comunicazione in tempo reale, perché rende percepibile lo sfasamento tra il labiale dell'interlocutore e l'audio elaborato.

Due precisazioni. Anzitutto il ritardo è di natura *puramente algoritmica*: non dipende dal tempo di calcolo sul dispositivo (la latenza computazionale, trattata nel Capitolo 7), ma unicamente dalla struttura della rete. Inoltre resta *invisibile* in addestramento e in valutazione offline, dove la rete elabora l'intera sequenza nella sua interezza e il *padding* simmetrico attinge a campioni futuri già disponibili. La dipendenza dal futuro si traduce in attesa soltanto nell'inferenza *streaming*: per questo le metriche offline del baseline restano buone, pur essendo il modello di fatto inutilizzabile dal vivo.

Nel privilegiare la qualità a scapito della causalità, la baseline rinuncia proprio alla proprietà real-time che il modello originale possedeva per costruzione. Il contributo architetturale di questa tesi consiste nel recuperarla senza alterare la struttura della rete, mediante una riformulazione *streaming* fondata sugli stati.

8.3 Riformulazione causale con gli stati

La convoluzione dilatata, cuore di ogni *ResBlock*, per calcolare l'uscita del frame t ha bisogno dei frame che lo precedono: con kernel di dimensione K e dilatazione d servono i frame da $t-(K-1)d$ fino a t. In streaming questi frame passati sono già stati elaborati e scartati: manca, semplicemente, la memoria del passato che la convoluzione richiede per funzionare.

Lo **stato** è esattamente questa memoria. Si tratta di un piccolo buffer che conserva gli ultimi frame entrati in un layer. L'idea è sostituire la dipendenza dai campioni futuri con il ricordo esplicito di quelli passati. Il calcolo eseguito resta identico a quello di una convoluzione causale, cambia soltanto il modo in cui i frame necessari vengono resi disponibili alla rete.

Conviene allora seguire il ciclo di vita di uno stato durante una singola inferenza, cioè durante l'elaborazione di un frame. Indichiamo con h_t il frame in ingresso alla convoluzione dilatata di un dato layer e con $\mathbf{s}_{t-1}$ lo stato lasciato dal passo precedente, un tensore che contiene gli ultimi $RF=(K-1)d$ frame. La rete compie tre operazioni in sequenza. Prima concatena passato e presente, formando una finestra di $RF+1$ frame, dopodiché applica la convoluzione, configurata senza *padding*, che consuma quella finestra e produce esattamente un frame di uscita ed, infine, aggiorna lo stato, conservando gli ultimi RF frame della finestra appena formata:

$$\mathbf{s}_t = \Big[\, \mathbf{s}_{t-1} \,\|\, h_t \,\Big]_{-RF:} \tag{8.2}$$

dove $\|$ denota la concatenazione lungo l'asse temporale e $[\,\cdot\,]_{-RF:}$ la selezione delle ultime RF colonne. L'effetto è quello di una finestra che scorre: a ogni frame lo stato sostituisce il campione più vecchio con quello appena arrivato, replicando un vero e proprio *buffer circolare*. Lo stato aggiornato $\mathbf{s}_t$ viene restituito insieme

alla maschera e reimmesso nella rete al frame successivo. È questo passaggio a chiudere il ciclo e a permettere alla rete di "ricordare".

Questa memoria non è globale e condivisa, infatti, ogni operazione di convoluzione ha il proprio stato privato, indipendente da quello degli altri layer. Il loro numero è perciò pari al prodotto tra la quantità di blocchi temporali e quelli dei layer per blocco ($n_{\text{blocks}} \times n_{\text{layers}}$), e ciascuno è un tensore di forma (B, C, RF), dove C è il numero di canali della scala ERB: poiché la dilatazione cresce come $d_i = 2^{i-1}$ ($i = 1, \ldots, n_{\text{layers}}$), gli stati hanno profondità diversa. Nella configurazione finale a 5 livelli per blocco, motivata nel Capitolo 9, la profondità va dai 2 frame del layer più superficiale ai 32 del più profondo. Più un layer guarda lontano nel tempo, più lunga è la memoria che deve trasportare. All'inizio della sequenza tutti gli stati sono posti a zero, il che equivale a dichiarare che, prima del primo frame, non c'è alcun passato da ricordare.

La rete, di per sé, non conserva nulla tra una chiamata e l'altra: riceve in ingresso il frame corrente e l'insieme degli stati, e restituisce la maschera e gli stati aggiornati. La memoria vive quindi *fuori* dal modello, nel codice che lo invoca e che ha il compito di rimettere in circolo gli stati a ogni passo.

8.4 Il modello finale

Il modello finale di questa tesi combina i tre contributi sviluppati nei capitoli precedenti in un'unica architettura causale, denominata ERB-TCNN-STREAMING. Il flusso di elaborazione, per ogni frame, è il seguente: la magnitudine spettrale viene compressa dal modulo ERB (Capitolo 7) nel dominio a 129 canali; i blocchi TCN causali la elaborano aggiornando i rispettivi stati; il modulo ERB riespande la maschera ai 257 bin originari, su cui agisce l'attivazione finale. La maschera è infine applicata allo spettro complesso rumoroso e il segnale è ricostruito via ISTFT.

L'addestramento adotta la Loud-loss percettiva, eventualmente combinata con il termine SI-SNR, descritta nel Capitolo 6. Coerentemente con il meccanismo a stati, a ogni *batch* gli stati sono inizializzati a zero e l'intera sequenza è processata in un'unica passata. In fase di valutazione, il percorso di inferenza esegue il **vero ciclo frame-by-frame**: a ogni passo la rete riceve un singolo frame e gli stati correnti e restituisce la maschera e gli stati aggiornati, confermando il funzionamento in tempo reale del modello esportato.

8.5 Dall'hardware embedded all'applicazione Android

L'obiettivo iniziale del progetto era l'esecuzione del modello su un microcontrollore **STM32N6**, scelto come unico "cervello" del sistema: la stessa scheda avrebbe dovuto occuparsi sia dell'acquisizione audio sia dell'inferenza real-time e del processamento del segnale. Durante lo sviluppo, tuttavia, è emerso un vincolo bloccante: i due processi risultavano *mutuamente esclusivi*. La scheda poteva acquisire l'audio oppure eseguire l'inferenza, ma l'esecuzione contemporanea delle due operazioni ne provocava il blocco. La medesima limitazione si è ripresentata anche utilizzando la scheda **STM32H7**. Ovviamente sono stati utilizzati tutti gli strumenti software messi a disposizione dall'azienda stessa.

Questo ostacolo — di natura sistemistica, indipendente dalla bontà del modello — ha reso impraticabile la soluzione puramente embedded inizialmente prevista. La risposta progettuale è stata orientarsi verso una piattaforma più accessibile e priva di hardware dedicato: un'**applicazione Android** eseguibile su un comune smartphone. Questa scelta non vanifica il lavoro architetturale svolto poiché il modello causale a stati è infatti *agnostico rispetto all'hardware*, e la sua natura streaming, frame-by-frame, si presta direttamente all'esecuzione su dispositivo mobile tramite ONNX Runtime. L'applicazione **SentoMeglio**, che concretizza questa soluzione, è descritta nel Capitolo 10.

Capitolo 9

Sperimentazione e selezione dei parametri del modello

L'architettura del modello è pronta, adesso manca discutere dei parametri e della STFT, in maniera sperimentale. Cronologicamente, questa sperimentazione ha accompagnato lo sviluppo del modulo ERB e della riformulazione causale dei Capitoli 7 e 8, che infatti adottano già i valori di `hop_length` e di profondità di rete qui selezionati: la si presenta solo ora perché è più chiaro seguire prima la logica costruttiva del modello, e solo dopo la sua validazione sperimentale.

9.1 Metodologia

La selezione dei parametri è, nella sua essenza, un problema a più obiettivi in conflitto tra loro. Da un lato c'è la qualità della voce ricostruita, dall'altro il costo che il modello impone al dispositivo quindi non esiste un valore "giusto" in assoluto, ma solo il miglior compromesso rispetto al contesto applicativo.

Per affrontare il problema in modo ordinato si è adottato un protocollo dove viene variato un solo fattore per volta. Si parte da una configurazione di riferimento, che chiameremo *punto operativo*, e si esplora un parametro alla volta facendo variare solo quello e tenendo fissi tutti gli altri.

I tre esperimenti che seguono sono **incrementali** ossia il primo individua l'ampiezza della finestra, il secondo, fissato il precedente valore, determina la profondità della rete e il terzo, trovati gli altri due, stabilisce il numero di blocchi. Quindi, ogni esperimento eredita dal precedente le scelte già consolidate e ne aggiunge una nuova, così che alla fine del percorso le tre decisioni convergano in un'unica configurazione, motivata in ogni sua parte. Il passo di avanzamento dell'STFT è invece fissato a 128 campioni — anziché ai 160 del riferimento STMicroelectronics

usato nei Capitoli 5 e 6 — e non rientra quindi tra i parametri da determinare sperimentalmente in questo capitolo.

- **Protocollo.** Tutti gli addestramenti condividono la stessa procedura, ereditata dal setup di riferimento dei capitoli precedenti, in modo che le differenze tra i modelli dipendano dal solo iperparametro in esame. Si addestra fino a 200 epoche con *early stopping* (pazienza di 15 epoche sulla metrica PESQ di validazione), ottimizzatore Adam con *learning rate* iniziale 10^{-3} regolato da uno scheduler `ReduceLROnPlateau`, e funzione di perdita $\mathcal{L}_{loud} + \mathcal{L}_{SI\text{-}SNR}$ nella variante risultata migliore nel Capitolo 6 (pesi $\alpha = 0{,}01$ e $\beta = 1{,}0$). La compressione ERB mantiene i suoi 65 canali lineari e 64 bande percettive.

- **Dataset.** L'addestramento e la valutazione si appoggiano al dataset VoiceBank+DEMAND, già impiegato nei capitoli precedenti. Va però chiarito un punto metodologico che incide direttamente sulla validità dei risultati. Questo dataset, a differenza di altri, non prevede una tripartizione canonica ma distingue soltanto un insieme di addestramento e uno di test, quest'ultimo costituito da parlatori che non compaiono mai nell'addestramento. Suddividere questo dataset in parti fisse avrebbe portato a problemi perché avrebbe quasi certamente distribuito le registrazioni di uno stesso parlatore su entrambi gli insiemi, lasciando filtrare informazione dal test all'addestramento e gonfiando le metriche. Si è perciò conservata la separazione per parlatore già prevista dal dataset, ricavando l'insieme di validazione da una piccola porzione dell'addestramento. È la scelta che garantisce sia l'assenza di contaminazione sia la confrontabilità con la letteratura.

- **Strumenti di valutazione.** Ogni modello è osservato su tre piani distinti e complementari. Il primo è quello delle metriche oggettive (PESQ, STOI, SNR, SI-SNR ed MSE), misurate sia per il modello in virgola mobile sia per la sua versione quantizzata. Il secondo è quello della valutazione percettiva non intrusiva tramite l'algoritmo DNSMOS, già descritto nei capitoli precedenti. Il terzo è quello del costo su dispositivo, profilato con lo strumento di analisi per il microcontrollore messo a disposizione da STM: occupazione di memoria, peso del modello e carico di calcolo.

- **Criterio di selezione.** Il criterio per la scelta finale del modello da utilizzare è definito dalla regola: a parità sostanziale di qualità, si preferisce la configurazione meno costosa e che si traduce nella ricerca del punto di intersezione della curva qualità-costo, cioè il punto oltre il quale spendere di più non porta alcun guadagno apprezzabile. Le differenze tra modelli vicini sono spesso minime, e per non scambiare per significativa una fluttuazione fortuita si è tenuta presente la *variabilità tra addestramenti*: due reti identiche, addestrate in

momenti diversi, non producono esattamente lo stesso risultato, e lo scarto che le separa fissa una soglia naturale sotto la quale ogni differenza va considerata rumore e non effetto.

9.2 Ampiezza della finestra di analisi

Il primo nodo da sciogliere riguarda la finestra dell'STFT. Mantenendo fissa la dimensione della trasformata, si è fatta variare l'ampiezza della finestra di analisi tra 256 e 512 campioni, lasciando immutato tutto il resto. La scelta tocca due aspetti che conviene tenere distinti. Il primo è la risoluzione tempo-frequenza: una finestra corta segue meglio le variazioni rapide del segnale ma distingue peggio le frequenze vicine, mentre una finestra lunga fa l'opposto. Il secondo è la latenza algoritmica di ricostruzione. Come si è visto nel Capitolo 8, la riformulazione causale a stati azzera il *look-ahead* della rete, e il passo di emissione *frame-by-frame* resta fissato dall'*hop*; ciò che ancora dipende dalla finestra è il ritardo con cui un campione viene *pienamente ricostruito* dalla sintesi *overlap-add*. La ricostruzione di un tratto di segnale, infatti, è definitiva solo dopo che tutte le finestre che lo coprono sono state raccolte e sommate, e l'ultima di esse si estende per l'intera durata della finestra di analisi. È un ritardo costante, fisicamente presente nel percorso dal microfono all'auricolare, e cresce in proporzione alla lunghezza della finestra.

Poiché la dimensione della trasformata resta invariata, il modello vede sempre lo stesso numero di canali in ingresso, per tale motivo l'occupazione di memoria e il carico di calcolo sono identici per ogni ampiezza di finestra e questo esperimento isola in modo netto il confronto tra qualità e latenza, senza interferenze legate al costo. La Tabella 9.1 riporta le metriche oggettive.

Formato	Finestra (latenza)	PESQ	STOI	SNR	SI-SNR	MSE ($\times 10^{-4}$)
float32	256 (16,0 ms)	2,633	0,939	17,480	17,374	1,257
	320 (20,0 ms)	2,686	0,941	18,074	17,992	1,112
	400 (25,0 ms)	2,679	0,945	18,377	18,316	1,044
	512 (32,0 ms)	**2,688**	**0,946**	**18,468**	**18,407**	**0,995**
int8	256 (16,0 ms)	2,572	0,938	16,633	16,722	1,507
	320 (20,0 ms)	2,649	0,940	17,278	17,357	1,319
	400 (25,0 ms)	2,608	**0,943**	**17,325**	**17,512**	**1,297**
	512 (32,0 ms)	**2,686**	**0,943**	16,736	16,961	1,476

Tabella 9.1: Metriche oggettive dell'esperimento che varia l'ampiezza della finestra

La lettura della tabella restituisce un andamento chiaro. La finestra più corta, 256 campioni, è la più debole su quasi ogni indice: la risoluzione in frequenza che sacrifica si traduce in una perdita misurabile di qualità, sia in virgola mobile sia dopo la

quantizzazione. Da 320 campioni in su, invece, la qualità entra in una fascia ristretta: in virgola mobile la finestra da 320 e quella da 512 sono praticamente sovrapposte sul PESQ (2,686 contro 2,688), con un lievissimo vantaggio di quest'ultima sugli indici energetici. Le metriche percettive sulle registrazioni reali, riportate nella Tabella 9.2, raccontano la stessa storia senza indicare un vincitore netto tra le finestre intermedie.

	Traffico (OVRL)		Folla (OVRL)	
Finestra	float32	int8	float32	int8
256	2,403	2,429	2,557	2,524
320	2,482	2,447	2,514	2,483
400	**2,554**	2,425	2,440	2,444
512	2,448	**2,462**	**2,683**	**2,603**

Tabella 9.2: Punteggio percettivo complessivo DNSMOS al variare dell'ampiezza della finestra.

A questo punto entra in gioco il criterio di selezione. Poiché il costo su dispositivo non cambia, la decisione si gioca tutta sull'equilibrio tra qualità e latenza di ricostruzione, e tecnicamente le metriche di pura qualità, prese da sole, indicherebbero la finestra da 512 come migliore parametro per la rete, ma quel vantaggio è minimo, mentre il prezzo in latenza è decisamente incisivo: 32 ms contro 20 ms. In un sistema pensato per l'ascolto dal vivo, dove la finestratura è ormai l'unica sorgente di latenza algoritmica, 12 millisecondi non sono un dettaglio ma sono margine che si propaga per tutto il resto della catena di elaborazione. La finestra da 320 campioni è la più corta che si colloca già sul plateau della qualità, recuperandola quasi per intero rispetto a quella della finestra lunga ma a metà del suo costo in termini di ritardo.

9.3 Profondità della rete

Fissata la dimensione della finestra a 320 campioni, il secondo esperimento interroga la profondità della rete, ovvero il numero di livelli che compongono ciascun blocco temporale. Si è fatto variare questo numero da tre a sette, lasciando inalterati il kernel e la progressione delle dilatazioni, in modo che l'unica differenza tra i modelli fosse la profondità. Qui il compromesso cambia natura rispetto all'esperimento precedente: aggiungere livelli non incide più sulla latenza ma allarga il campo ricettivo della rete, ossia la quantità di passato che essa è in grado di osservare, e al tempo stesso ne aumenta il costo. Conviene quindi guardare come si muovono qualità e risorse.

Il campo ricettivo, riportato nella colonna "contesto" della Tabella 9.3, si ricava dalla struttura della rete. All'interno di ogni blocco le dilatazioni dei livelli crescono

come potenze di due $(1, 2, 4, \ldots, 2^{L-1})$, per cui il numero di frame osservati da una rete con B blocchi di L livelli ciascuno e kernel di ampiezza K vale

$$R = 1 + B\,(K-1)\,(2^L - 1). \tag{9.1}$$

Con i valori adottati ($B = 2$ blocchi, $K = 3$) il coefficiente $B\,(K-1)$ è pari a 4. Il contesto in secondi si ottiene moltiplicando R per la durata di un frame, fissata dal passo dell'STFT: 128 campioni a 16 kHz, cioè 8 ms. Così, ad esempio, cinque livelli danno $R = 1 + 4\,(2^5 - 1) = 125$ frame, pari a circa un secondo di segnale. Poiché il termine dominante è 2^L, ogni livello aggiunto **raddoppia** all'incirca il contesto osservato, ed è questo il meccanismo che, come si vedrà, governa anche la crescita della memoria.

Formato	Livelli (contesto)	PESQ	STOI	SNR	SI-SNR	MSE ($\times 10^{-4}$)
float32	3 (0,23 s)	2,625	0,938	17,801	17,709	1,214
	4 (0,49 s)	2,661	0,941	18,015	17,934	1,144
	5 (1,00 s)	2,685	**0,943**	18,118	18,048	1,107
	6 (2,02 s)	**2,688**	**0,943**	**18,142**	**18,065**	1,093
	7 (4,07 s)	2,665	0,942	18,073	17,997	**1,089**
int8	3 (0,23 s)	2,574	0,937	16,876	16,931	1,473
	4 (0,49 s)	2,631	0,940	17,038	17,253	1,371
	5 (1,00 s)	**2,640**	0,942	**17,354**	**17,455**	1,315
	6 (2,02 s)	2,629	**0,943**	17,254	17,346	**1,291**
	7 (4,07 s)	2,631	0,941	16,922	17,090	1,373

Tabella 9.3: Metriche oggettive per l'esperimento che vaaria il numero di livelli per blocco.

La Tabella 9.3 mostra una crescita netta della qualità da tre a cinque livelli, con un guadagno di PESQ ad ogni gradino che resiste anche alla quantizzazione. Oltre il quinto livello, però, il quadro si fa ambiguo e va letto con attenzione, evitando di fermarsi al valore numericamente più alto. In virgola mobile è il sesto livello a esibire i numeri migliori su quasi tutti gli indici, ma si tratta di scarti dell'ordine del millesimo: il vantaggio di PESQ del sesto sul quinto è così piccolo da rientrare nella normale oscillazione tra due addestramenti, e non costituisce quindi un miglioramento reale. Nella versione quantizzata la gerarchia addirittura si rovescia, dato che il quinto livello è il migliore sul PESQ e l'altro peggiora in modo statisticamente significativo. Il settimo livello, infine, degrada in entrambi i formati: la rete più profonda diventa più difficile da addestrare e da quantizzare senza che il contesto aggiuntivo porti alcun beneficio. Il confronto tra i due livelli, insomma, non premia nettamente nessuno, e lo stesso accade sulle registrazioni reali della Tabella 9.4, dove il formato *float32* favorisce il sesto livello anche se con una differenza poco sostanziale. Quindi questi due livelli, in qualità, vanno considerati sostanzialmente equivalenti.

	Traffico (OVRL)		Folla (OVRL)	
Livelli	float32	int8	float32	int8
3	2,409	2,395	2,530	2,393
4	2,491	2,405	2,540	2,480
5	2,542	**2,471**	2,611	2,529
6	**2,652**	2,414	**2,690**	**2,602**
7	2,549	2,422	2,657	2,599

Tabella 9.4: Punteggio percettivo complessivo DNSMOS al variare del numero di livelli.

La saturazione ha una spiegazione fisica. Con cinque livelli il campo ricettivo della rete copre già circa un secondo di segnale, un intervallo ampiamente sufficiente a cogliere la struttura temporale del parlato; portarlo a due o quattro secondi, come fanno il sesto e il settimo livello, significa offrire alla rete un contesto che per il compito di denoising è in larga parte ridondante. Dal lato del costo, però, la spesa è tutt'altro che ridondante, ed è qui che si trova l'argomento decisivo. La Tabella 9.5 riassume l'occupazione di risorse sul microcontrollore di destinazione.

Livelli	Parametri	FLASH float32	FLASH int8	RAM float32	RAM int8
3	228.138	891,5 KiB	221,6 KiB	30,3 KiB	8,2 KiB
4	295.992	1156,6 KiB	287,5 KiB	62,5 KiB	18,5 KiB
5	363.846	1421,8 KiB	353,3 KiB	127,0 KiB	36,6 KiB
6	431.700	1687,0 KiB	419,2 KiB	256,0 KiB	72,9 KiB
7	499.554	1952,1 KiB	485,0 KiB	514,0 KiB	145,5 KiB

Tabella 9.5: Costo su dispositivo al variare del numero di livelli.

Il dato che salta all'occhio è il comportamento della memoria di lavoro, che **raddoppia** a ogni livello aggiunto, mentre il peso del modello cresce solo in modo lineare. La ragione sta nella struttura della rete: l'occupazione di memoria è dominata dallo stato del livello più profondo, la cui lunghezza è determinata dalla dilatazione, e poiché le dilatazioni crescono come potenze di due, ogni livello in più raddoppia la profondità dello stato più costoso. È lo stesso meccanismo descritto nel Capitolo 8, dove gli stati spaziavano da pochi frame a diverse decine: aggiungere profondità significa pagare un prezzo che cresce in modo esponenziale.

A questo punto la decisione spetta al costo. Se quinto e sesto livello sono equivalenti in qualità, la scelta ricade naturalmente sul più economico, e qui il divario è netto: il sesto livello richiede il doppio della memoria di lavoro del quinto. Pagare un raddoppio di RAM per un vantaggio che non supera il rumore di addestramento sarebbe ingiustificato. Il quinto livello è perciò il punto di intersezione tra qualità e costo e diventa la profondità adottata.

9.4 Numero di blocchi temporali

Fissati i risultati precedenti, resta da decidere quanti blocchi TCN impilare. Si è fatto variare il loro numero da uno a tre. Anche qui il parametro agisce sul campo ricettivo e sul costo, ma in un modo che, come si vedrà, è strutturalmente diverso da quello dei livelli. La Tabella 9.6 riporta le metriche oggettive.

Formato	Blocchi (contesto)	PESQ	STOI	SNR	SI-SNR	MSE ($\times 10^{-4}$)
float32	1 (0,50 s)	2,616	0,940	17,997	17,904	1,163
	2 (1,00 s)	**2,680**	**0,942**	18,086	18,009	1,118
	3 (1,50 s)	2,672	0,941	**18,143**	**18,078**	**1,085**
int8	1 (0,50 s)	2,540	0,940	**17,267**	17,253	1,386
	2 (1,00 s)	**2,618**	**0,941**	17,111	**17,269**	1,347
	3 (1,50 s)	2,607	0,940	17,037	17,231	**1,341**

Tabella 9.6: Metriche oggettive al variare del numero di blocchi TCN.

Il primo fatto da registrare è che sul test esteso le tre configurazioni si somigliano parecchio. Sulle metriche energetiche un solo blocco tiene il passo degli altri due: in *float32* lo SNR si sposta di appena un decimo di decibel da una configurazione all'altra, e in *int8* il modello a un blocco segna perfino il valore di SNR più alto della colonna, pareggiando su STOI e SI-SNR. L'unico indice su cui un blocco resta davvero indietro è il PESQ, dove cede circa sei centesimi in virgola mobile e otto dopo la quantizzazione: uno scarto misurato, non enorme, ma abbastanza sopra la soglia di rumore da non poter passare per fluttuazione. Diverso è il confronto tra due e tre blocchi, che si consuma in pochi millesimi, ed è proprio qui che la cautela metodologica annunciata in apertura diventa indispensabile. Si dà il caso che la configurazione a due blocchi coincida con quella già addestrata nell'esperimento precedente: disponendo così di due addestramenti distinti della stessa identica rete, è possibile misurare direttamente quanto due esecuzioni differiscano per effetto del solo caso. Lo scarto tra questi due addestramenti gemelli è dell'ordine di un paio di centesimi di PESQ, cioè *maggiore* della distanza che separa due blocchi da tre. La conclusione è netta: tra due e tre blocchi la differenza è indistinguibile dal rumore di addestramento, e i due modelli vanno considerati equivalenti in qualità.

Le metriche percettive sulle registrazioni reali, nella Tabella 9.7, danno una vista più ordinata. Il punteggio cresce in modo regolare con il numero di blocchi, da uno a tre, quasi senza eccezioni nelle quattro colonne di traffico e folla, ed è l'andamento più pulito che si sia visto in tutto l'esperimento; preso alla lettera indicherebbe tre blocchi come la scelta migliore, va però soppesato al prezzo dettato dal costo.

Infatti, quest'ultimo rimane come sempre l'ago della bilancia. La Tabella 9.8 mostra che il numero di blocchi fa crescere le risorse in modo **lineare**. Tre blocchi costano di più di due, sia in peso sia in memoria, a fronte di un guadagno di qualità che si è dimostrato relativo.

	Traffico (OVRL)		Folla (OVRL)	
Blocchi	float32	int8	float32	int8
1	2,384	2,411	2,448	2,402
2	2,548	2,437	2,583	**2,587**
3	**2,696**	**2,570**	**2,661**	2,564

Tabella 9.7: Punteggio percettivo complessivo DNSMOS al variare del numero di blocchi.

Blocchi	Parametri	FLASH int8	RAM int8	RAM float32	Operazioni (int8)
1	194.211	188,7 KiB	20,1 KiB	65,0 KiB	226k
2	363.846	353,3 KiB	36,6 KiB	127,0 KiB	422k
3	533.481	517,9 KiB	50,3 KiB	173,4 KiB	619k

Tabella 9.8: Costo su dispositivo al variare del numero di blocchi.

La scelta finale ricade quindi su due blocchi temporali, ed è una decisione di equilibrio prima che di qualità.

9.5 Generalizzazione al mondo reale

I tre esperimenti precedenti convergono su una configurazione precisa, ma tutti sono stati condotti su un unico dataset, relativamente contenuto e registrato in condizioni controllate. Un sistema destinato all'uso quotidiano incontra però una varietà di rumori enormemente più ampia e imprevedibile di quella che un dataset di addestramento, per quanto curato, può racchiudere. Per colmare questa distanza si è condotta una seconda campagna di addestramento, indipendente dalla precedente, sulla stessa configurazione selezionata ma su un dataset di tutt'altra scala, costruito a partire dalla raccolta della DNS Challenge: un insieme vastissimo di voci pulite e di rumori reali, in larga parte non stazionari, dai quali vengono sintetizzate le miscele di addestramento mescolando parlato e disturbo a rapporti segnale-rumore variabili. A differenza del dataset precedente, questa raccolta prevede già un insieme di sviluppo separato, riservato alla valutazione e tenuto rigorosamente fuori dall'addestramento, così che la separazione tra ciò che la rete vede e ciò su cui viene giudicata sia garantita per costruzione.

La valutazione oggettiva di questo nuovo modello è stata condotta su questo insieme di test, ben più arduo e variegato di quello impiegato fin qui; i suoi valori assoluti, riportati nella Tabella 9.9, non vanno quindi confrontati con quelli delle sezioni precedenti, misurati in condizioni più benevole, ma letti come la prestazione del modello nel dominio realistico per cui è stato addestrato.

Il valore vero di questo secondo addestramento si apprezza però sulle registrazioni reali, dove la metrica percettiva non intrusiva può confrontare il modello allenato

Formato	PESQ	STOI	SNR	SI-SNR	MSE ($\times 10^{-4}$)
float32	2,098	0,894	13,812	13,290	1,591
int8	1,896	0,882	11,464	10,964	3,182

Tabella 9.9: Metriche oggettive del modello riaddestrato sul dataset DNS Challenge: non direttamente confrontabili con le tabelle precedenti, ottenute su un test set diverso e più semplice.

su DNS Challenge con quello addestrato sul solo dataset iniziale, a parità di architettura e di clip. La Tabella 9.10 mette a fianco le due versioni in virgola mobile, scomponendo il giudizio nelle sue componenti: qualità complessiva, naturalezza della voce e soppressione del rumore di fondo.

Scenario	Modello	OVRL	SIG	BAK	P808_MOS
Traffico	dataset iniziale	2,548	**3,030**	3,401	2,853
	DNS Challenge	**2,639**	2,963	**3,747**	**2,960**
Folla	dataset iniziale	**2,583**	**2,973**	**3,625**	**2,979**
	DNS Challenge	2,425	2,790	3,580	2,931

Tabella 9.10: Comparazione metriche soggettive tra il modello addestrato sul solo dataset iniziale e quello addestrato su DNS Challenge, sulle due clip reali di test (formato *float32*).

Il quadro che emerge è coerente con la natura dei due dataset. Sullo scenario di traffico, dominato da un rumore di fondo pervasivo, l'addestramento su DNS Challenge porta un miglioramento marcato proprio nella soppressione del fondo (l'indice BAK sale da 3,401 a 3,747), e con esso crescono la qualità complessiva e la naturalezza percepita. Sullo scenario di folla, fatto di voci sovrapposte e quindi di un disturbo che assomiglia al parlato stesso, il modello allenato su DNS Challenge cede qualcosa sulla componente vocale, perché l'esposizione a un rumore così aggressivo lo rende più cauto nel preservare ogni dettaglio della voce. Il bilancio netto, però, è quello atteso: addestrare su una varietà di rumori incomparabilmente più ricca produce un sistema più solido di fronte all'imprevedibilità del mondo reale, e in particolare più efficace nel cancellare i disturbi non stazionari, che sono esattamente quelli con cui l'applicazione finale dovrà confrontarsi.

9.6 Scelta del formato

Rimane un'ultima decisione, quella tra il modello in virgola mobile e la sua versione quantizzata a 8 bit. Lungo tutti gli esperimenti il formato in virgola mobile si è mostrato sistematicamente superiore in qualità, mentre la quantizzazione comporta una perdita contenuta ma costante. La scelta tra i due non è però soltanto

una questione di qualità, ma dipende in modo determinante dalla piattaforma di destinazione. Se il sistema dovesse girare sul microcontrollore inizialmente previsto, la quantizzazione sarebbe obbligatoria, infatti, l'acceleratore neurale di quel dispositivo lavora esclusivamente in aritmetica intera.

Tuttavia, come si è argomentato nel Capitolo 8, il vincolo sistemistico incontrato sul microcontrollore ha spostato il bersaglio verso un'applicazione per dispositivi *mobile*, dove il modello viene eseguito tramite un motore di inferenza ONNX che gestisce nativamente e con efficienza l'aritmetica in virgola mobile. Cade così l'obbligo della quantizzazione, ed entrambi i formati tornano percorribili. Potendo scegliere, si adotta allora il modello in *float32* allenato su DNS Challenge, perché è quello che esprime la massima qualità raggiungibile mantenendo il funzionamento in tempo reale: è la combinazione che meglio rimuove la maggior parte dei rumori non stazionari nelle condizioni d'uso effettive, ed è la versione che sarà impiegata nell'applicazione descritta di seguito.

Capitolo 10

SentoMeglio

Questo capitolo tratta lo sviluppo dell'applicazione *mobile* necessaria per eseguire la pipeline di *speech enhancement* in tempo reale su smartphone. Il suo obiettivo è rendere accessibile tale tecnologia alle persone senza richiedere configurazioni tecniche, con un'esperienza d'uso immediata.

10.1 Architettura generale dell'applicazione

L'applicazione è strutturata attorno a una separazione netta tra due livelli con responsabilità distinte, come illustrato in Figura 10.1.

- **Livello Kotlin.** Gestisce tutto ciò che riguarda il framework Android: il ciclo di vita delle Activity, la renderizzazione dell'interfaccia utente, la richiesta e la verifica dei permessi e altri servizi di sistema. Questo livello gira sulla *Java Virtual Machine* (JVM), usufruisce del *garbage collector* per la gestione automatica della memoria e può interagire con le API di alto livello.
- **Livello C++.** Gestisce tutto ciò che richiede accesso diretto all'hardware: la cattura e la riproduzione audio, l'intera pipeline di elaborazione DSP, l'inferenza neurale e le strutture dati per la sincronizzazione tra thread paralleli. Questo livello è compilato con l'NDK di Android per l'architettura ARM del dispositivo e gira direttamente sulla CPU senza l'overhead della JVM.

I due livelli sono connessi tramite interfaccia JNI (*Java Native Interface*), il meccanismo standard che permette al codice Kotlin di invocare funzioni C++ e di riceverne i risultati. Questa è deliberatamente minimale, ossia il livello Kotlin non conosce i dettagli basso livello e, al contrario, il livello C++ non conosce i dettagli dell'altro, quindi ognuno lavora esclusivamente nel proprio dominio.

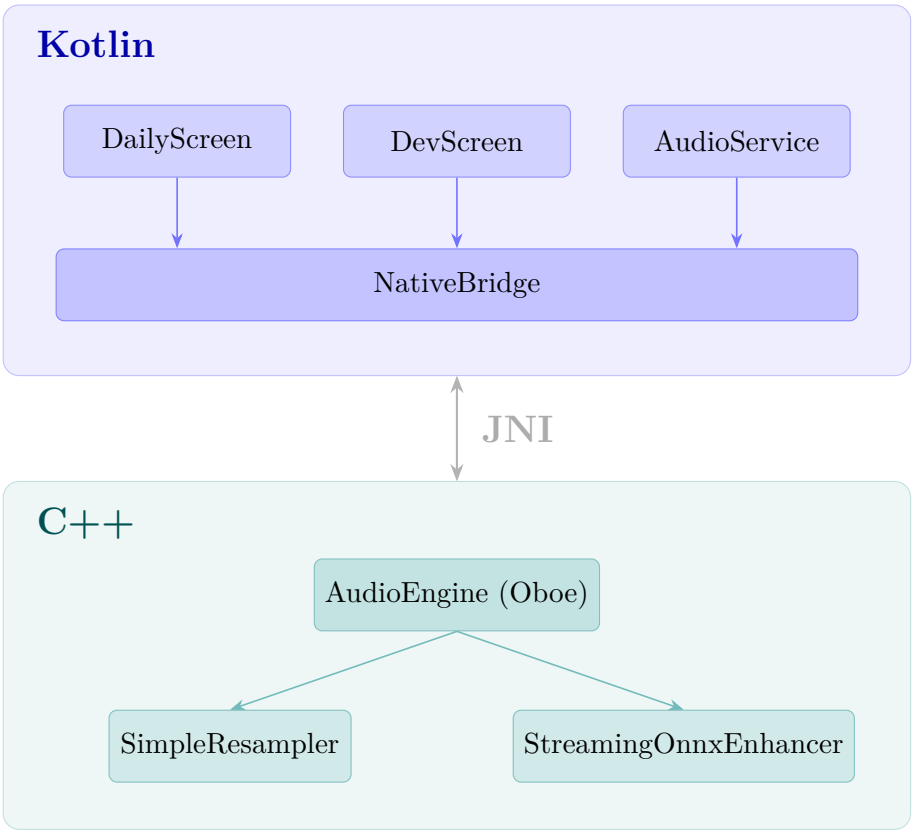


Figura 10.1: Architettura dell'applicazione ad alto livello: il livello Kotlin (interfaccia utente) e quello C++ (acquisizione audio, DSP e inferenza) comunicano attraverso l'interfaccia JNI.

10.2 Interfaccia utente

L'applicazione prevede due schermate distinte, selezionabili tramite una pagina di impostazioni, la schermata *daily*, pensata per l'utente finale, e la schermata *developer*, progettata per avere dettagli tecnici e per configurare i parametri.

10.2.1 Schermata daily

Questa *activity*, mostrata in Figura 10.2, è progettata rispettando i principi di accessibilità, infatti, esiste un solo tasto interattivo, nessuna terminologia tecnica, nessun parametro da configurare. Il design tiene conto che il target primario sono le persone con difficoltà uditive, tra cui molti anziani, per i quali un'interfaccia complessa sarebbe una barriera d'uso.

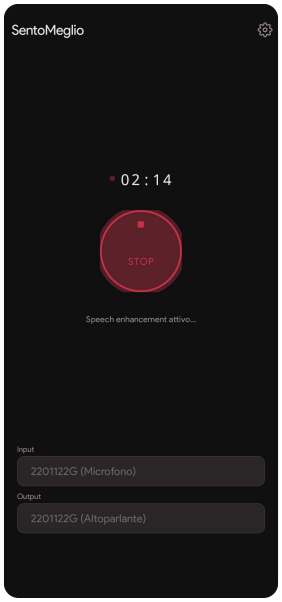


Figura 10.2: Schermata *daily*.

Alla pressione del tasto **REC**, avvia la pipeline audio e cambia aspetto visivo; una seconda pressione ferma l'elaborazione. Un timer nel formato `HH:MM` fornisce un riferimento temporale sulla durata della sessione.

Due spinner collocati nella parte inferiore della schermata consentono di selezionare il dispositivo di **ingresso** e di **uscita**. I dispositivi visualizzati sono filtrati in base alla compatibilità con la pipeline: microfono interno, cuffie cablate, cuffie Bluetooth SCO e BLE.

10.2.2 Schermata developer

L'*activity* in questione, mostrata in Figura 10.3, è uno strumento di diagnostica. Permette di cambiare alcuni dei parametri ed espone in tempo reale tutte le metriche rilevanti per la valutazione delle prestazioni del sistema:

- **Parametri STFT**: campi di testo modificabili per `n_fft`, `hop_length` e `win_length`.
- **Metriche di latenza**: latenza hardware separata tra ingresso e uscita (misurata direttamente da Oboe), latenza DSP, latenza di inferenza, RTF, contatore di XRun, dimensione del buffer e del burst, modalità di condivisione (Exclusive o Shared).
- **Spettrogrammi**: due grafici affiancati che mostrano in tempo reale lo spettrogramma del segnale rumoroso in ingresso e di quello pulito in uscita. Ogni vista

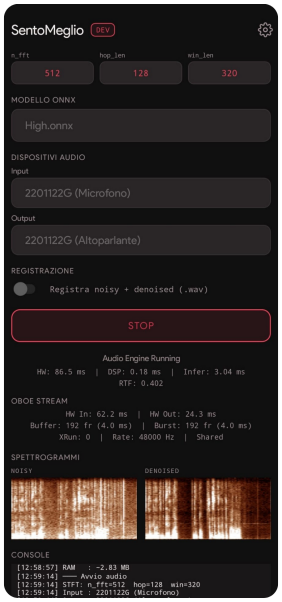


Figura 10.3: Schermata *developer*.

visualizza 100 frame temporali per 257 bin di frequenza con una colormap di tipo magma. La visualizzazione scorre orizzontalmente a ogni hop, mostrando l'evoluzione spettrale in corso.

- **Terminale dei log**: un pannello scrollabile con le ultime 150 righe che mostrano i log dell'applicazione in tempo reale.
- **Registrazione audio**: un toggle che abilita la scrittura su file del segnale rumoroso e del segnale pulito. Se abilitato, permette di riascoltare i file audio in un secondo momento.

10.3 Cattura e riproduzione audio: la libreria Oboe

L'acquisizione e la riproduzione audio sono gestite dalla libreria **Oboe** [24], una libreria C++ open-source di Google progettata per applicazioni audio a bassa latenza su Android. Essa funge da contenitore che astrae la frammentazione hardware e software del sistema operativo, espone un'interfaccia unica che utilizza nativamente le API moderne a bassissima latenza *AAudio* (disponibili da Android 8.0) ed esegue un fallback del tutto trasparente sulla libreria legacy *OpenSL ES* sui dispositivi più datati, garantendo prestazioni ottimali e massima retrocompatibilità.

10.3.1 Configurazione delle stream

Vengono aperte due stream Oboe simultanee: una di ingresso (acquisizione dal microfono) e una di uscita (riproduzione).

Entrambe sono configurate con formato *float32*, mono (un canale), e la frequenza di campionamento nativa del dispositivo, tipicamente 48 kHz. Quest'ultima è tale per cui l'hardware audio è ottimizzato, quindi richiedere una frequenza diversa obbligherebbe il driver a effettuare una conversione interna, il che aumenterebbe la latenza, per questo motivo il *resampling* viene gestito esplicitamente nel software, come descritto nella Sezione 10.4.

La **modalità di prestazione** è impostata su `LowLatency`: questa opzione informa Android che la stream ha requisiti di latenza stringenti, il che spinge il sistema ad allocare buffer di dimensione minima e ad assegnare al thread audio una priorità di scheduling elevata.

La **modalità di condivisione** richiesta per le stream audio è `Exclusive`. Quì il flusso tenta di ottenere l'accesso esclusivo a un *endpoint* del dispositivo, aggirando il mixer software di sistema per interfacciarsi direttamente con il buffer elaborato dal DSP. Questo approccio fornisce la minor latenza possibile. Tuttavia, tale esclusività non è sempre garantita, infatti, ci sono smartphone che proprio non erogano questa possibilità mentre, qualora fosse disponibile, la richiesta può essere respinta se la risorsa è già in uso. In questi scenari, oppure quando si utilizzano periferiche non supportate, Oboe effettua automaticamente un *fallback* alla modalità `Shared`.

10.3.2 Il callback audio

Oboe gestisce il flusso dati tramite un **callback** invocato periodicamente dal sistema operativo su un thread ad alta priorità con scheduling real-time. Il callback viene chiamato ogni *burst* di campioni e deve completare entro il budget temporale corrispondente. Se supera il budget, si verifica un **XRun** (dall'inglese *under-run* o *over-run*): un glitch udibile causato dall'interruzione del flusso audio.

Per rispettare il vincolo real-time, il callback deve soddisfare due regole fondamentali: non effettuare mai allocazioni dinamiche di memoria (che coinvolgono il sistema operativo e il garbage collector) e non acquisire mai mutex (che potrebbero causare attesa indeterminata).

10.4 La pipeline di elaborazione audio

L'elaborazione vera e propria del segnale è distribuita su due thread con compiti nettamente separati e disaccoppiati da code *lock-free*. Il primo è il **thread audio** dove risiede, appunto, Oboe che si limita a scambiare campioni con l'hardware; il secondo è un **thread di inferenza** dedicato, che esegue l'intera catena DSP

e la rete neurale. Questa separazione nasce da un vincolo preciso, infatti, una singola inferenza richiede alcune decine di millisecondi, un tempo incompatibile con il budget real-time del callback, ragion per cui il carico pesante non può risiedere nel thread audio. La Figura 10.4 riassume il percorso del segnale, dal microfono fino alla riproduzione.

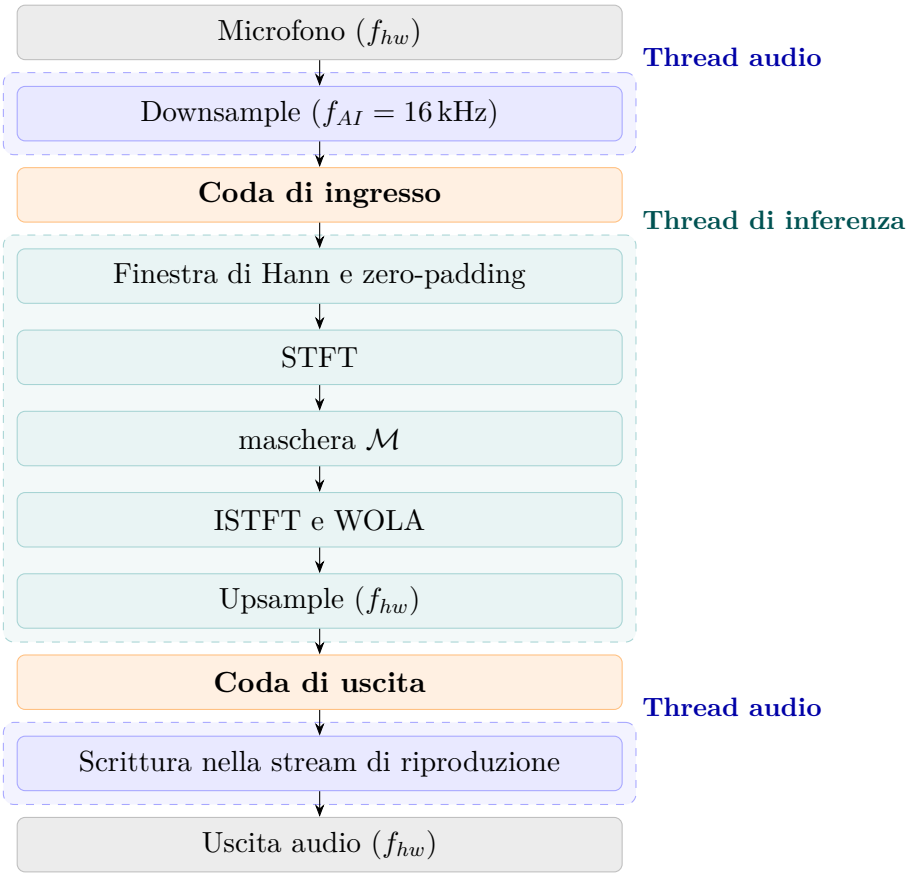


Figura 10.4: Percorso del segnale dal microfono alla riproduzione: agli estremi le fasi eseguite dal thread audio, al centro quelle del thread di inferenza, disaccoppiati da code *lock-free*.

10.4.1 Il flusso tra i due thread

Il thread audio si limita a quattro operazioni elementari, cioè legge un *burst* dal microfono alla frequenza nativa f_{hw}, lo sottocampiona a $f_{AI} = 16\,\text{kHz}$ e lo deposita nella coda di ingresso, invece nella direzione opposta, preleva dalla coda di uscita i campioni già elaborati e li scrive nella stream di riproduzione. Tutto il carico

computazionale resta così confinato nel thread di inferenza, quest'ultimo consuma la coda di ingresso non appena vi si è accumulato almeno un *hop* di campioni.

Le due conversioni, inoltre, non risiedono nello stesso thread. Il *downsample* da f_{hw} a f_{AI} avviene all'ingresso, dentro il thread audio, mentre l'*upsample* inverso avviene all'uscita, dentro l'altro thread. In questo modo entrambe le code trasportano campioni già pronti per il loro destinatario, quindi il callback non deve mai ricampionare e resta sempre entro il proprio budget. Il ricampionamento è affidato alla libreria Speex [25], configurata con un filtro *sinc* polifase di qualità intermedia, un compromesso fra fedeltà della conversione e latenza introdotta.

10.4.2 Estrazione della magnitudine

Il thread di inferenza accumula i campioni in arrivo in un buffer circolare lungo quanto la finestra di analisi (`win_length` = 320 campioni). La struttura circolare evita di spostare fisicamente i dati a ogni passo, infatti, a ogni *hop* i 128 campioni nuovi sovrascrivono i 128 più vecchi e l'indice di scrittura avanza di modulo `win_length` e il contenuto del buffer viene poi riletto in ordine cronologico per formare il frame da analizzare.

Il frame di 320 campioni viene quindi pesato campione per campione con una finestra di Hann:

$$w[n] = \frac{1}{2}\left(1 - \cos\!\left(\frac{2\pi n}{N_w - 1}\right)\right), \quad n = 0, \ldots, N_w - 1 \tag{10.1}$$

dove $N_w = 320$. Tale finestra attenua gradualmente i campioni verso i bordi (valori prossimi a zero agli estremi, massimo al centro), ammorbidendo la discontinuità che altrimenti si crea quando il segnale termina bruscamente. Questo riduce lo *spectral leakage*, il fenomeno per cui l'energia si disperderebbe su tutte le frequenze mascherando la struttura armonica del parlato.

Sul frame così preparato si calcola la trasformata di Fourier reale (**RFFT**) con la libreria **pocketfft** [26], un'implementazione C++ header-only dell'algoritmo di Cooley-Tukey. Da ciascun bin complesso si estraggono magnitudine e fase:

$$|X(m,k)| = \sqrt{\Re\{X(m,k)\}^2 + \Im\{X(m,k)\}^2}, \tag{10.2}$$

$$\angle X(m,k) = \operatorname{atan2}(\Im\{X(m,k)\}, \Re\{X(m,k)\}) \tag{10.3}$$

Al modello viene passata la sola magnitudine, mentre la fase viene conservata e riutilizzata nella ricostruzione del segnale.

10.4.3 Maschera e risintesi

Il vettore delle 257 magnitudini viene fornito all'ONNX Runtime [27] come tensore di forma $[1, 257, 1]$. Poiché il modello esportato è la versione *streaming* del Capitolo 8, l'inferenza riceve e restituisce anche gli stati ricorrenti che ne costituiscono la memoria temporale, ragion per cui a ogni *hop* il codice si limita a ricopiarli dalle uscite agli ingressi, secondo il meccanismo a stati descritto nella Sezione 8.3. È questo riporto a garantire la continuità tra frame consecutivi.

L'uscita di interesse è la maschera spettrale $\mathcal{M}(m, k) \in [0,1]$, applicata secondo l'approccio di *magnitude masking*: riscala bin per bin la magnitudine rumorosa:

$$|\hat{S}(m, k)| = |X(m, k)| \cdot \mathcal{M}(m, k) \tag{10.4}$$

La fase, invece, non viene toccata, ma si riutilizza quella del segnale rumoroso. Lo spettro pulito stimato in forma complessa $\hat{S}(m, k) = |\hat{S}(m, k)|\, e^{j\angle X(m,k)}$ viene quindi riportato nel dominio del tempo dalla trasformata inversa (**IRFFT**).

I frame ricostruiti si sovrappongono per il 60 %, ragion per cui vanno ricombinati con il metodo **Weighted Overlap-Add** (WOLA). Ogni frame viene pesato con la stessa finestra di Hann impiegata in analisi e accumulato in un buffer di sintesi, normalizzato dalla somma corrente dei quadrati della finestra:

$$\hat{s}[n] = \frac{\sum_{m} w_s[n - mH] \cdot x_m^{\mathrm{IRFFT}}[n - mH]}{\sum_{m} w_s^2[n - mH]} \tag{10.5}$$

con $H =$ `hop_length` e $w_s[n] = w[n]$. Il denominatore compensa la modulazione di ampiezza introdotta dalla doppia finestratura, inoltre, poiché la normalizzazione è esplicita, a regime la ricostruzione è esatta quando la maschera è unitaria ($\mathcal{M} = 1$), indipendentemente da quanto bene la finestra soddisfi la condizione di overlap-add (ossia che la somma dei suoi valori, traslata di un hop alla volta, rimanga costante per ogni campione). Dal buffer di sintesi si estraggono infine i primi 128 campioni, che costituiscono l'uscita del frame corrente, mentre il buffer scorre poi di altrettante posizioni in attesa del frame successivo.

Questi campioni a 16 kHz vengono sovracampionati a f_{hw}, dallo stesso thread in cui si trova, e scritti nella coda di uscita, dalla quale il callback Oboe li preleva per la riproduzione, senza ulteriori conversioni.

10.5 Strategie adottate per minimazzare la latenza

La latenza end-to-end è il parametro progettuale più critico per un ausilio uditivo in tempo reale. La si può scomporre in tre contributi: latenza hardware, algoritmica

e di elaborazione, ognuno dei quali richiede strategie di ottimizzazione proprie.

10.5.1 Latenza hardware

Questo è il ritardo introdotto dall'hardware audio del dispositivo, dalle operazioni del sistema operativo di Android e dalla libreria Oboe stessa. Include la latenza della stream di ingresso (dal momento in cui i campioni arrivano al microfono fino alla loro disponibilità al software) e quella in uscita (dal software verso la periferica di output).

Per minimizzare questa componente si adottano due strategie:

1. **Modalità Exclusive**: come descritto nella Sezione 10.3, l'accesso esclusivo all'hardware elimina il mixer software di Android, che aggiungerebbe un ritardo fisso legato al ciclo di mixing.

2. **Buffer minimo**: la dimensione del buffer di riproduzione viene impostata al valore minimo supportato dall'hardware (*burst size*). Un buffer più piccolo significa che i campioni vengono consumati più frequentemente, riducendo il tempo di attesa tra la scrittura software e la riproduzione fisica.

Questa latenza viene misurata in tempo reale dalla libreria Oboe tramite metodi dedicati e visualizzata nella schermata developer. Il suo valore dipende fortemente dallo smartphone e dalla periferica audio selezionata: per il microfono interno e gli altoparlanti integrati è tipicamente di poche decine di ms, al contrario delle cuffie Bluetooth, dove si può arrivare a valori significativamente più elevati.

10.5.2 Latenza algoritmica

Qui il ritardo è intrinseco dell'algoritmo, indipendente dalla velocità con cui lo si esegue. Nel caso della STFT dipende da due fattori, cioè dall'eventuale uso di campioni futuri e dalla lunghezza della finestra di analisi.

Il primo fattore è già stato neutralizzato a monte. La scelta `center=False` (Capitolo 4) fa sì che ogni frame dipenda solo da campioni presenti e passati, mentre la riformulazione causale della rete (Capitolo 8) ne elimina la dipendenza dai frame futuri, ragion per cui il *look-ahead* complessivo è nullo.

Resta il secondo fattore, ed è qui che conviene distinguere due ritardi. Il primo è la **latenza di ricostruzione**: la sintesi *overlap-add* fornisce il valore *definitivo* di un campione solo quando tutte le finestre che lo coprono sono state sommate, e poiché l'ultima di queste termina circa `win_length` campioni dopo, il ritardo con cui un campione è pienamente ricostruito è dell'ordine della finestra di analisi:

$$\tau_{\text{alg}} \approx \frac{\texttt{win_length}}{f_{AI}} = \frac{320}{16000} = 20\,\text{ms} \tag{10.6}$$

È questa la latenza algoritmica in senso stretto, quella usata come riferimento nel Capitolo 9, ed è determinata dalla finestra, non dall'*hop*. Il secondo ritardo è il **passo di emissione**: la pipeline lavora *frame-by-frame* e, grazie all'inizializzazione a zero degli stati e del buffer di analisi, restituisce un nuovo blocco di campioni a ogni `hop_length`, senza attendere il riempimento dell'intera finestra. Da questo discende l'intervallo

$$T_{\text{hop}} = \frac{\texttt{hop_length}}{f_{AI}} = \frac{128}{16000} = 8\,\text{ms} \tag{10.7}$$

che non è un ritardo di percorso, ma la cadenza con cui il sistema produce uscita. È anche il **budget temporale** entro cui l'intera pipeline (STFT, inferenza e ISTFT) deve completare un frame, pena l'impossibilità di stare al passo con il flusso in ingresso, e sarà il riferimento per il monitoraggio in tempo reale discusso più avanti.

10.5.3 Thread e code

La separazione tra thread audio e thread di inferenza, già introdotta nella Sezione 10.4, nasce proprio da un'esigenza di latenza. Il callback dispone di pochi millisecondi per ogni *burst*, mentre una singola inferenza ne richiede molti di più, ragion per cui eseguirla al suo interno ne provocherebbe il superamento sistematico, con XRun continui. Affidarla a un thread separato scioglie il vincolo, perché i due lavorano con regimi temporali diversi:

- il **thread audio** ha priorità real-time e non può mai bloccarsi né eccedere il proprio budget, infatti, si occupa soltanto di leggere e scrivere sulle stream hardware, operazioni dal costo costante e prevedibile;
- il **thread di inferenza** ha priorità elevata ma non real-time, quindi può tollerare oscillazioni nella durata di esecuzione (*jitter*) senza ripercuotersi sul thread audio, purché in media resti entro il budget.

A disaccoppiare i due thread sono le due code che li collegano. Queste ultime sono dimensionate con ampio margine, così da assorbire le fluttuazioni temporali senza mai saturarsi. La coda di uscita, inoltre, viene pre-riempita all'avvio con un breve tratto di silenzio, in modo da coprire il ritardo dettato dalla prima inferenza senza emettere campioni non validi.

Il requisito cruciale è che il thread audio non resti mai in attesa dell'altro, ecco perché le code sono *lock-free* ossia ciascuna ammette un solo produttore e un solo consumatore e ognuno aggiorna esclusivamente la propria posizione di lettura o di scrittura, mentre un contatore condiviso, aggiornato in modo atomico, segnala quanti campioni sono disponibili. In questo modo nessuno dei due thread deve acquisire un *lock* né rischia di rimanere bloccato dall'altro.

10.5.4 RTF ed EMA

Per sapere istante dopo istante se il sistema sta tenendo il passo, si ricorre al **Real-Time Factor** (RTF), che confronta il tempo speso a elaborare un frame con il tempo che il frame stesso impiega ad arrivare:

$$RTF = \frac{t_{\text{DSP}} + t_{\text{inf}}}{T_{\text{hop}}} \tag{10.8}$$

dove t_{DSP} e t_{inf} sono i tempi di elaborazione DSP e di inferenza, mentre T_{hop} è il budget di 8 ms ricavato in precedenza. La lettura è immediata, cioè un RTF inferiore a 1 significa che l'elaborazione è più rapida del flusso in ingresso — la condizione necessaria al funzionamento in tempo reale — mentre al contrario segnala un sistema in deficit.

I tempi misurati oscillano da un frame all'altro per via della variabilità del sistema operativo, per cui vengono attenuati con una **media mobile esponenziale** (EMA):

$$\hat{t}^{(m)} = \alpha \cdot t^{(m)} + (1 - \alpha) \cdot \hat{t}^{(m-1)}, \quad \alpha = 0.1 \tag{10.9}$$

Con $\alpha = 0.1$, ogni nuova misura contribuisce per un decimo e i nove decimi rimanenti provengono dalla stima precedente; i picchi isolati, come quelli causati dalla sospensione temporanea del thread da parte del sistema operativo, vengono così smorzati senza alterare l'andamento medio.

10.6 Esecuzione in background

Perché SentoMeglio sia utile come ausilio uditivo, deve poter operare ininterrottamente anche quando l'utente non sta guardando lo schermo del telefono: durante una conversazione, colui che sta usando l'applicazione, ha l'attenzione rivolta all'interlocutore, non al dispositivo. Questo requisito si scontra con le politiche di gestione del ciclo di vita di Android, che sono progettate per preservare la durata della batteria riducendo le risorse allocate alle applicazioni in background.

10.6.1 Android Foreground Service

Android gestisce le applicazioni secondo un modello gerarchico in cui le risorse vengono reclamate in ordine di priorità. Quando l'utente manda un'applicazione in background, il sistema operativo può:

1. Fermare l'Activity (deallocando le risorse UI ma mantenendo il processo)
2. Fermare il processo (richiamando `onStop` e `onDestroy`)

3. Terminare il processo senza preavviso per recuperare memoria (*low memory kill*)

Un'applicazione ordinaria non può resistere a queste politiche: se il suo processo viene terminato, tutta l'elaborazione audio si interrompe. La soluzione è dichiarare che l'applicazione ha un'attività critica in corso tramite un **Foreground Service**, ossia un componente che segnala al sistema operativo come il proprio processo svolga un'attività avviata consapevolmente dall'utente, che non deve essere interrotta silenziosamente. In cambio di questa garanzia di continuità, Android impone che il servizio mostri una **notifica persistente** nell'area di notifica — un segnale visibile che ricorda all'utente che l'applicazione è in esecuzione e sta consumando risorse.

Capitolo 11

Conclusioni

11.1 Limiti del lavoro

Il risultato è un sistema completo e funzionante, ma alcuni limiti vanno dichiarati con chiarezza.

Il primo è di natura sistemistica. L'intero sistema avrebbe dovuto risiedere su un microcontrollore STM32N6. L'obiettivo embedded originario, quindi, non è stato raggiunto, e lo smartphone è subentrato come piattaforma alternativa.

Il secondo riguarda la valutazione. Le metriche di qualità e intelligibilità sono state calcolate offline, mentre sul dispositivo le latenze e il RTF sono soltanto monitorati in tempo reale dalla schermata diagnostica, senza una campagna di misura sistematica su un parco di telefoni eterogeneo. Manca inoltre la prova più importante per un ausilio uditivo, ovvero una valutazione soggettiva con utenti ipoacusici reali, che resta l'unico vero giudice dell'utilità pratica del sistema.

Restano, infine, alcuni limiti legati al funzionamento sul campo: la latenza complessiva dipende fortemente dalla periferica audio, e le cuffie Bluetooth, costrette alla modalità *Shared*, introducono ritardi sensibilmente più alti del percorso cablato. La fase, inoltre, viene riutilizzata rumorosa, scelta che semplifica la pipeline ma pone un tetto noto alla qualità ricostruibile.

11.2 Sviluppi futuri

I limiti appena elencati tracciano in modo naturale le direzioni di lavoro successive.

La più urgente è una validazione sul campo, cioè una caratterizzazione sistematica della latenza end-to-end e del consumo su più dispositivi, seguita da uno studio percettivo con utenti ipoacusici che misuri il guadagno reale di intelligibilità in ambienti rumorosi.

Sulla parte modellistica, due strade appaiono promettenti. La prima è il superamento dell'approccio a fase rumorosa, tramite la sua stima esplicita o un modello che operi direttamente nel dominio complesso. La seconda è la personalizzazione sull'audiogramma dell'utente, così da modellare l'elaborazione sulla specifica perdita uditiva anziché applicare una soppressione uniforme.

Un altro aspetto da poter migliorare riguarda la selettività degli interlocutori: si tratterebbe dielevare il modello a una maggiore risoluzione spaziale, passando da un singolo microfono ad un array. Questo salto in avanti permetterebbe di sentire la voce pulita anche di persone che non si trovano dinanzi all'utente, rendendo di fatto l'esperienza più naturale possibile.

Infine, c'è la possibilità di un ritorno all'hardware dedicato su dispositivi embedded.

Acronimi

BAK
Background intrusiveness

BLE
Bluetooth Low Energy

CNN
Convolutional Neural Network

DFT
Discrete Fourier Transform

DL
Deep Learning

DSP
Digital Signal Processing

EMA
Exponential Moving Average

ERB
Equivalent Rectangular Bandwidth

FFT
Fast Fourier Transform

FNN
Feedforward Neural Network

GPU

Graphics Processing Unit

GRU

Gated Recurrent Unit

IA

Intelligenza Artificiale

IRM

Ideal Ratio Mask

ISTFT

Inverse Short-Time Fourier Transform

JNI

Java Native Interface

JVM

Java Virtual Machine

Leaky ReLU

Leaky Rectified Linear Unit

LSTM

Long Short-Term Memory

ML

Machine Learning

MOS

Mean Opinion Score

MSE

Mean Squared Error

NDK

Native Development Kit

ONNX

Open Neural Network Exchange

OVRL

Overall quality

PESQ

Perceptual Evaluation of Speech Quality

PReLU

Parametric Rectified Linear Unit

ReLU

Rectified Linear Unit

ELU

Exponential Linear Unit

RNA

Reti neurali artificiali

RNN

Recurrent Neural Network

RTF

Real-Time Factor

SCO

Synchronous Connection-Oriented

SI-SNR

Scale Invariant Signal-to-Noise Ratio

SIG

Signal distortion

SNR

Signal-to-Noise Ratio

SPL

Sound Pressure Level

STFT

Short-Time Fourier Transform

STOI

Short-Time Objective Intelligibility

TCN

Temporal Convolutional Networks

TCNN

Temporal Convolutional Neural Network

WOLA

Weighted Overlap-Add

Bibliografia

[1] Mojisola A Bolarinwa e Majirioghene E Ayoma. «Development of a Signal Processing Algorithm for Feedback (Noise) Reduction in Hearing Aids Using the Spectral Subtraction Technique». In: *European Journal of Electrical Engineering and Computer Science* 8.3 (2024), pp. 21–28 (cit. a p. 2).

[2] Caroline Catherman. *The next wave of hearing aids (Spoiler alert: It's AI!)* 2025. URL: https://www.healthcare-brew.com/stories/2025/03/26/next-wave-hearing-aids-ai (cit. a p. 2).

[3] Dulani Meedeniya. *Deep learning: A beginners' guide*. Chapman e Hall/CRC, 2023 (cit. a p. 5).

[4] Roheen Qamar e Baqar Ali Zardari. «Artificial neural networks: An overview». In: *Mesopotamian Journal of Computer Science* 2023 (2023), pp. 124–133 (cit. a p. 7).

[5] Aravind Pai. *12 Types of Neural Networks in Deep Learning*. 2025. URL: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/ (cit. a p. 11).

[6] Wikipedia. *Speech Enhancement*. 2024. URL: https://en.wikipedia.org/wiki/Speech_enhancement (cit. a p. 12).

[7] Speechprocessing. *Speech Enhancement*. URL: https://speechprocessingbook.aalto.fi/Speech_enhancement.html (cit. a p. 13).

[8] DeLiang Wang e Jitong Chen. «Supervised speech separation based on deep learning: An overview». In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.10 (2018), pp. 1702–1726 (cit. alle pp. 13, 14).

[9] Microsoft. *Speech Enhancement*. URL: https://www.microsoft.com/en-us/research/project/nn-speech-enhancement/ (cit. a p. 14).

[10] Ravikumar Kandagatla, V. Jayachandra Naidu, P.S. Sreenivasa Reddy, Gayathri K M, Jahnavi A e Rajeswari K. «Analysis of statistical estimators and neural network approaches for speech enhancement». In: *SciEnggJ* (2024). URL: https://api.semanticscholar.org/CorpusID:268350739 (cit. a p. 16).

[11] Microsoft. *DNS-Challenge*. URL: https://github.com/microsoft/dns-challenge (cit. a p. 16).

[12] Soha A Nossier, Julie Wall, Mansour Moniri, Cornelius Glackin e Nigel Cannings. «A comparative study of time and frequency domain approaches to deep learning based speech enhancement». In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8 (cit. alle pp. 17, 18).

[13] Kuldip Paliwal e Kamil Wójcicki. «Effect of analysis window duration on speech intelligibility». In: *IEEE signal processing letters* 15 (2008), pp. 785–788 (cit. a p. 18).

[14] Jonathan Allen. «Short term spectral analysis, synthesis, and modification by discrete Fourier transform». In: *IEEE transactions on acoustics, speech, and signal processing* 25.3 (2003), pp. 235–238 (cit. a p. 18).

[15] Ashutosh Pandey e DeLiang Wang. «TCNN: Temporal Convolutional Neural Network for Real-time Speech Enhancement in the Time Domain». In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 6875–6879. DOI: 10.1109/ICASSP.2019.8683634 (cit. alle pp. 21, 45).

[16] STMicroelectronics. *STFT-TCNN*. URL: https://github.com/STMicroelectronics/stm32ai-modelzoo/blob/main/speech_enhancement/stft_tcnn/README.md (cit. alle pp. 21, 29, 34, 35, 43, 44).

[17] Zixuan Li, Xueliang Zhang, Changjiang Zhao, Shuai Gao, Lei Miao, Zhipeng Yan, Ying Sun e Chong Zhu. «Loud-loss: A Perceptually Motivated Loss Function for Speech Enhancement Based on Equal-Loudness Contours». In: *arXiv preprint arXiv:2511.05945* (2025) (cit. alle pp. 30, 31, 35).

[18] *Acoustics — Normal equal-loudness-level contours*. Standard ISO 226:2023. Geneva, Switzerland: International Organization for Standardization, 2023. URL: https://www.iso.org/standard/83117.html (cit. a p. 30).

[19] Cassia Valentini-Botinhao. *Noisy speech database for training speech enhancement algorithms and TTS models*. 2017. DOI: 10.7488/ds/2117. URL: https://doi.org/10.7488/ds/2117 (cit. a p. 34).

[20] Brian R Glasberg e Brian CJ Moore. «Derivation of auditory filter shapes from notched-noise data». In: *Hearing research* 47.1-2 (1990), pp. 103–138 (cit. a p. 38).

[21] Jean-Marc Valin, Umut Isik, Neerad Phansalkar, Ritwik Giri, Karim Helwani e Arvindh Krishnaswamy. «A perceptually-motivated approach for low-complexity, real-time enhancement of fullband speech». In: *arXiv preprint arXiv:2008.04259* (2020) (cit. a p. 39).

[22] Xiaobin Rong, Tianchi Sun, Xu Zhang, Yuxiang Hu, Changbao Zhu e Jing Lu. «GTCRN: A speech enhancement model requiring ultralow computational resources». In: *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2024, pp. 971–975 (cit. a p. 39).

[23] Shaojie Bai, J Zico Kolter e Vladlen Koltun. «An empirical evaluation of generic convolutional and recurrent networks for sequence modeling». In: *arXiv preprint arXiv:1803.01271* (2018) (cit. a p. 45).

[24] Google. *Oboe is a C++ library that makes it easy to build high-performance audio apps on Android.* 2025. URL: `https://github.com/google/oboe` (cit. a p. 63).

[25] Xiph.Org Foundation. *Speex: a free codec for free speech — SpeexDSP resampler library.* 2022. URL: `https://www.speex.org` (cit. a p. 66).

[26] Martin Reinecke e Peter Bell. *pocketfft: header-only C++ FFT library.* 2024. URL: `https://github.com/mreineck/pocketfft` (cit. a p. 66).

[27] ONNX Runtime developers. *ONNX Runtime: cross-platform, high-performance ML inferencing and training accelerator.* 2024. URL: `https://onnxruntime.ai` (cit. a p. 67).