

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA



CORSO DI LAUREA TRIENNALE IN INFORMATICA

**SPEAKER RECOGNITION SU RETI NEURALI: IL PESO
DEGLI I-VECTOR**

Relatore:

PROF. ROCCO ZACCAGNINO

PROF. ROBERTO DE PRISCO

Candidato:

MICHELE IANNUCCI

MAT. 0512105725

ANNO ACCADEMICO 2020-2021

INDICE

INDICE	3
INTRODUZIONE	5
CAPITOLO 1 BACKGROUND TEORICO	9
1.1 Rete Siamese	9
1.2 Deep Learning, Reti Neurali e CNN	11
1.2.1 Deep Learning	12
1.2.2 Rete Neurale	12
1.2.3 CNN	13
1.3 x-vector	14
1.4 i-vector	16
1.5 Embedding fusion	19
CAPITOLO 2 RELATED WORK	20
CAPITOLO 3 MODIFICHE APPORTATE	23
3.1 Configurazione estrazione i-vector	25
3.2 Estrazione combinata	26
3.3 Estrazione multiprocessing	30
3.4 Webapp	37
3.5 Front-end ed utilizzo	40
3.6 Back-end	44
CAPITOLO 4 TESTING E SPERIMENTAZIONE	48
CONCLUSIONI	50
BIBLIOGRAFIA	52

INTRODUZIONE

Il seguente lavoro di tesi è incentrato su una delle branche dell'Intelligenza Artificiale che sta godendo di maggiore crescita in termini di interesse e risultati in questi anni, lo speaker recognition.

Speaker Recognition sta ad indicare l'attività di verifica dell'identità dichiarata dalla persona che sta parlando, che avviene mediante l'analisi delle caratteristiche della sua voce.

Spesso si parla indistintamente di riconoscimento dell'impronta vocale, proprio perché si tratta di una tecnologia di autenticazione biometrica.

Esattamente come le impronte digitali, la voce è una caratteristica biologicamente univoca di ogni persona.

Ognuno di noi ha il proprio modo di parlare, il proprio accento così come il proprio ritmo, e tali differenze sono dovute anche alle caratteristiche fisiche, come il tratto vocale o gli organi che producono il suono, la dimensione della laringe e così via.

La difficoltà principale riguarda l'estrazione e la definizione di parametri che possano riflettere al meglio le caratteristiche di chi sta parlando, al fine di creare un buon modello di riconoscimento.

In questo senso, la ricerca ha raggiunto buoni livelli di modellazione, grazie ai paradigmi che si sono susseguiti in questi anni.

Tutte le nuove declinazioni di questo campo condividono alla base il medesimo processo decisionale, organizzato in tre grosse fasi:

1. Feature extraction: la voce viene analizzata e da essa si ricavano i tratti rilevanti utili al riconoscimento
2. Confronto delle feature estratte dalla voce che sta parlando con le feature salvate in memoria relative alla reale voce del parlante, tramite un modello di verifica
3. Esito del riconoscimento sulla base dei risultati ottenuti, in termini di somiglianza delle suddette caratteristiche

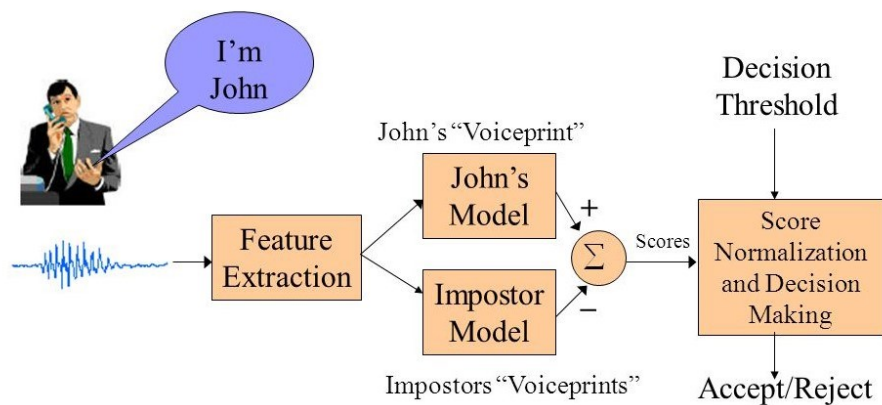


Figura 0-1: Speaker Recognition process

Tale ambito ha catturato fin da subito la mia attenzione, per due motivi. Innanzitutto perché la voce è il principale metodo di comunicazione che adottiamo, e sarebbe quindi molto interessante poter utilizzarla anche per interagire con i computer.

Il secondo motivo, connesso al primo, riguarda le potenzialità che ha questa tecnologia di diventare parte integrante della nostra vita.

Infatti sebbene la ricerca stia facendo passi da gigante, questi sistemi non sono ancora utilizzati, specie in situazioni critiche, a causa dell'accuratezza non ancora soddisfacente.

Le declinazioni di utilizzo potrebbero essere le più disparate, dall'adozione dell'autenticazione vocale come alternativa agli odierni sistemi biometrici basati su impronta o sul viso per l'utilizzo dei propri device, laptop o autovetture. Oppure come ulteriore prova di autenticazione per autorizzare un pagamento o per accedere ad aree sensibili, come il proprio home-banking.

Attualmente infatti utilizziamo la nostra voce in un ambito tecnologico differente dallo speaker recognition. Quando utilizziamo dispositivi di traduzione istantanea per comunicare con qualcuno che parla una lingua diversa, siamo nel caso di Language Recognition.

Quando invece, più comunemente, interroghiamo gli assistenti dei nostri dispositivi di domotica casalinga, ricadiamo nel caso di Speech Recognition. Gli assistenti ci rispondono una volta che hanno capito le parole da noi pronunciate.

Lo speaker recognition invece traslascia le parole o la lingua utilizzata, perché è interessato a capire chi è che sta parlando, non cosa sta dicendo.

Esistono poi altre due branche correlate: lo Speaker Identification che riconosce il parlante confrontandolo non solo con la sua voce nel database, ma anche con molte altre differenti voci, e la Speaker Diarization che si occupa di distinguere le varie persone che parlano in una traccia, e riconoscerle tutte correttamente.

L'obiettivo di questo lavoro di tesi sarà quello di continuare il lavoro di un progetto esistente che descriveremo a breve, migliorandone non solo l'aspetto grafico, ma soprattutto l'accuratezza del riconoscimento in un particolare contesto.

Il prossimo capitolo fornirà le conoscenze basilari per affrontare la lettura, nel capitolo 2 verrà illustrato il progetto da cui siamo partiti. Successivamente saranno presentate tutte le modifiche apportate ed in ultimo saranno evidenziati e discussi i risultati ottenuti.

Capitolo 1

BACKGROUND TEORICO

In questa sezione forniremo al lettore gli aspetti teorici basilari degli argomenti trattati, al fine di facilitarne la comprensione. Spiegheremo quindi cos'è una Rete Siamese, motore dell'intero progetto, ed in cosa consistono gli x-vector ed i-vector che verranno spesso nominati in questo elaborato di tesi.

1.1 Rete Siamese

Una Rete Siamese si dice tale proprio perché eredita il concetto di “gemelli siamesi” dalla biologia.

Essa è infatti una rete neurale composta da due sottoreti gemelle, quindi totalmente uguali, sia per ciò che concerne la struttura, sia per l'assegnazione dei pesi w .

Proprio per la natura del funzionamento che illustreremo tra un attimo, esse hanno trovato una vasta applicazione nel campo del riconoscimento biometrico, sia facciale che vocale.

Infatti sia che si tratti di un'immagine raffigurante un viso, oppure di un file audio contenente una voce, la rete effettua una comparazione tra

due oggetti: quello in input, ad esempio il viso rilevato in quell'istante, e quello conservato nel database al momento della prima registrazione.

I due oggetti vengono passati in modo distinto alle due reti gemelle che lavorano in parallelo. Ognuna delle due attua una fase detta di incorporamento (embedding) in cui ottiene una rappresentazione vettoriale dell'input.

Geograficamente ci troviamo nel Embedding Layer Representation, livello all'interno del quale la rete funge anche da feature extractor, per estrarre i suddetti vettori.

Una volta ottenuti i due vettori corrispondenti, la rete utilizza dei Comparison Layers per calcolare la cosine distance o distanza euclidea, cioè una sorta di indice di somiglianza dei due input.

A questo punto se si tratta di tratti biometrici della stessa persona, l'indice di somiglianza dei due vettori confrontati dovrebbe quasi coincidere, e quindi avere una distanza molto bassa in termini numerici.

Al contrario, se stiamo confrontando la rappresentazione vettoriale di due persone diverse, la distanza è molto maggiore.

Generalmente infatti si interpreta la distanza dei due vettori in input nell'intervallo $(0,1)$, grazie all'applicazione di una sigmoid function.

Su tale risultato viene poi applicata una funzione di loss per aggiornare i pesi della rete, che coincidono nelle due sottoreti.

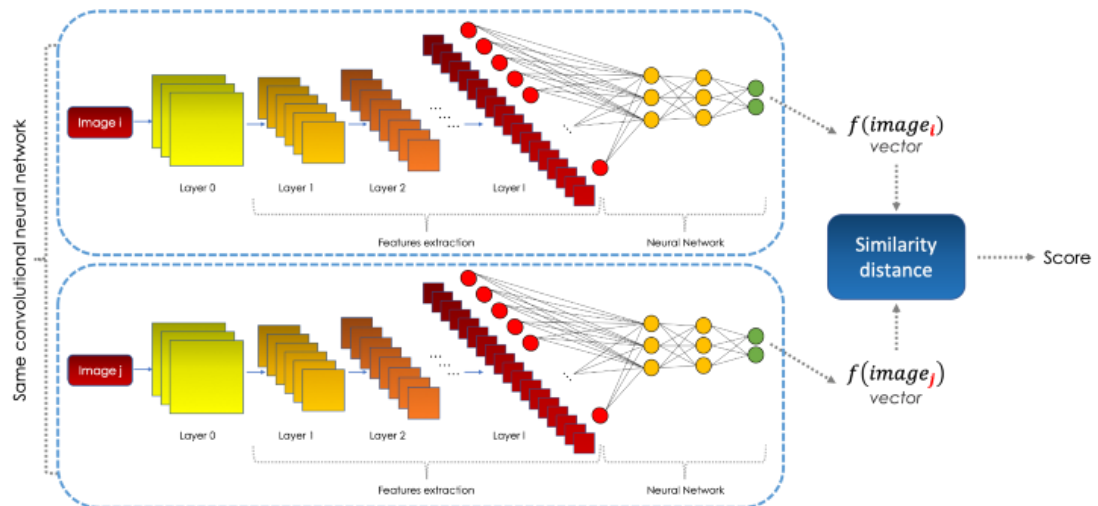


Figura 1-1: Raffigurazione di una Rete Siamese

Com'è mostrato chiaramente in figura, i due input vengono passati alle due reti gemelle, le quali fanno feature extraction per ottenere un vettore. I due vettori vengono confrontati e si ottiene una distanza nell'intervallo 0-1. Lo score ottenuto tipicamente si rapporta ad un valore soglia, per cui se la distanza è minore del threshold value allora il riconoscimento biometrico ha esito positivo, altrimenti negativo.

1.2 Deep Learning, Reti Neurali e CNN

In questo paragrafo intendiamo fornire una panoramica generale del contesto in cui ci troviamo. Le reti siamesi infatti non sono altro che applicazioni di reti neurali convolutive, CNN, operanti nell'ambito del deep learning.

1.2.1 *Deep Learning*

Il **deep learning** è la branca del machine learning che si serve di architetture multilivello, NN, per applicare algoritmi di apprendimento automatico nel campo della computer vision, nel riconoscimento delle immagini, nella bioinformatica e nello speaker recognition.

Nel machine learning vengono estratte le informazioni salienti per creare dei classificatori. Ma tali criteri sono selezionati a priori.

Nel deep learning invece l'estrazione delle feature avviene in modo autonomo, quindi è la rete che impara ad analizzare tutti i dati grezzi, riuscendo a determinarne le caratteristiche principali.

Questo avviene proprio perché le reti hanno a disposizione diversi livelli per l'apprendimento di caratteristiche via via più complesse.

Risulta quindi ovvio pensare che quanti più livelli intermedi ha a disposizione la rete, tanto più è capace di ottenere risultati migliori nella differenziazione delle caratteristiche.

1.2.2 *Rete Neurale*

Una **rete neurale** NN è una rete di neuroni artificiali organizzata su tre strati: il primo è il livello di input, segue un livello nascosto e poi quello di output.

Il livello nascosto è interamente connesso sia al livello di input che di output.

Nel caso in cui l'hidden layer è a sua volta costituito da almeno 2 livelli, si parla di rete profonda, deep (DNN).

1.2.3 CNN

Tra le reti più utilizzate nell'ambito del deep learning troviamo le CNN, reti neurali convolutive.

La loro nascita risale agli anni 90 grazie a Yann Le Cun, che le presentò come un'architettura preposta alla classificazione di immagini.

Il modello delle **CNN** è stato ispirato dalla scoperta relativa alle modalità con cui la corteccia visiva del nostro cervello opera durante il riconoscimento degli oggetti.

Essa è infatti costituita da layer o finestre differenti che via via estraggono le caratteristiche dell'oggetto, partendo dalla forma, colore fino ad arrivare ai dettagli minori.

La loro caratteristica principale sta nel fatto che vengono strutturati dei rilevatori di caratteristiche su più livelli.

Per creare una tale mappa di caratteristiche, si utilizza una sorta di meccanismo a finestre sovrapposte, che si fanno scorrere sull'input.

Tale processo di estrazione delle caratteristiche è detto convoluzione, da cui deriva il nome attribuito al tipo di rete.

Se ad esempio diamo in input un'immagine, il processo di estrazione delle caratteristiche si basa sull'utilizzo di diverse finestre o layer che estrapolano le feature partendo da un livello di dettaglio minore. Vengono prima esaminate le feature di basso livello, come i bordi, e poi si passa a quelle più complesse, come le forme, i contorni, i colori e così via.

Strutturalmente le CNN hanno quindi diversi convolutional layer, seguiti da pooling layers, tutti interamente connessi tra loro.

Le rete siamese del nostro progetto utilizza come sottoreti le reti CNN, le quali estraggono le caratteristiche, e la rete siamese non fa altro che calcolare la distanza tra i due risultati in output.

1.3 x-vector

Nell'ambito dello speaker recognition recentemente (nel 2018) è diventato uno standard l'utilizzo delle Deep Neural Network DNN, le quali vengono allenate per estrarre feature caratteristiche utili allo scopo di riconoscere chi sta parlando, e superano in termini di risultato il framework degli i-vector, che erano lo stato dell'arte fino a quel momento.

Tali feature sono denominate x-vector, ed hanno l'abilità di rappresentare espressioni vocali di lunghezza variabile in embeddings o vettori a dimensione fissa.

Durante la fase di training, le DNN hanno diversi layers utili ad estrarre informazioni rilevanti per il riconoscimento dello speaker.

In particolare, uno degli hidden layer, il segment6, rappresenta l'x-vector, cioè un vettore delle caratteristiche relative alla traccia vocale in input.

Per estrarre gli embeddings dello speaker, gli x-vector, la DNN utilizza come input delle features MFCC a 30 dimensioni, estratte da segnali audio di 25 millisecondi e normalizzati mediante l'utilizzo di una finestra scorrevole di circa 3 secondi. I segmenti privi di audio vengono intercettati e scartati grazie ad un processo di VAD, Voice Activity Detection basato sul segnale audio.

Generalmente, la DNN è organizzata su 3 livelli, frame, statistics e segment level.

Il primo livello **frame-level** è composto da 5 layers, grazie ai quali i frame audio vengono presi in input e splittati con una finestra scorrevole di 3 secondi, e dati in input alla TDNN, Time Delay Neural Network.

Poiché l'audio è un segnale di sequenza temporale, l'informazione è differente in ogni momento, ed attraverso la TDNN la rete apprende le informazioni strutturali del segnale e la relazione che c'è tra i vari frame.

Tale meccanismo di apprendimento è molto importante per il buon esito del riconoscimento vocale, poiché le caratteristiche della pronuncia e della cadenza di ogni persona possono variare molto all'interno di una sequenza temporale, e quindi riuscire a determinare una correlazione tra le varie sequenze diventa un aspetto cruciale.

Lo **statistic-layer** gioca un ruolo essenziale perché converte un segnale audio di lunghezza variabile in un vettore a dimensione fissa.

Per far ciò si serve di un solo layer, denominato statistics-pooling, il quale calcola deviazione media e standard.

Il **segment-level** mappa l' x-vector ottenuto all'identità dello speaker, dando in output la probabilità che si tratti di quello speaker.

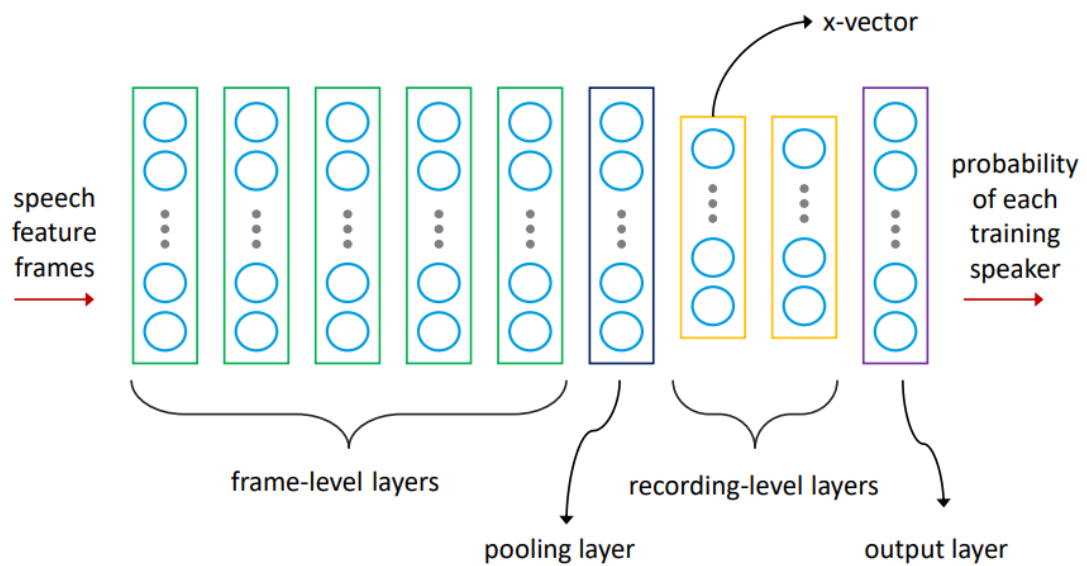


Figura 1-2: architettura della DNN

In generale il successo degli x-vector è dettato dai risultati ottenuti in termini di efficacia ed accuratezza, specie per file audio di breve durata, rispetto alle performance degli i-vector.

1.4 i-vector

Nel campo del riconoscimento vocale si sono succeduti diversi modelli nel corso del tempo, che hanno migliorato le performance dei predecessori, imponendosi come il riferimento da utilizzare per chiunque svolgesse ricerca in quest'area.

Il primo modello varato fu GMM, Gaussian Mixture Models nel 1990. Dieci anni più tardi ci fu il primo miglioramento, un adattamento chiamato GMM-UBM.

Nel 2018 si è giunti allo stato dell'arte col modello degli i-vector.

La maggior parte dei sistemi di autenticazione vocale è infatti attualmente basata sull'utilizzo degli i-vector.

Generalmente si utilizzano l'universal background model UBM per ottenere statistiche ed una matrice di large projection T che le mappa in una rappresentazione a bassa dimensione, che prende il nome di i-vector.

Tale rappresentazione è un singolo sottospazio che contiene tutte le caratteristiche dello speaker e la variabilità tra le varie sessioni.

Si applica poi una probabilistic linear discriminant analysis (PLDA) per comparare gli i-vector, e decidere se corrispondono allo stesso speaker oppure no.

Il vantaggio principale degli i-vector (che sfrutteremo nel nostro progetto) è che essi non sono strettamente dipendenti dal cambio del canale di trasmissione o dalla variabilità delle caratteristiche vocali dello speaker (come cadenza o accento), poiché nel modello proposto da Dehak basato sulla Joint Factor Analysis si tiene conto di entrambi i fattori nel complesso, durante la fase di modellazione.

Il modello JFA elimina infatti la distinzione tra i sottospazi della variabilità dello speaker e del canale, modellandoli in unico sottospazio a bassa dimensione, detto Total variability space.

Questo sottospazio si può rappresentare come:

$$s = m + Tw$$

Dove m è la componente indipendente dal parlante e dal canale, T è la matrice che copre la variabilità sia dello speaker che del canale e w è un vettore casuale.

Per ciò che riguarda il processo di estrazione, la figura seguente mostra le due parti coinvolte: front-end and back-end.

Nella prima vengono estratte le features, mentre nella seconda vengono estratti e classificati gli i-vector.

Siccome abbiamo visto che gli i-vector modellano anche l'informazione relativa alla variazione del canale, talvolta vengono attuate tecniche di channel compensation come Linear Discriminant Analysis (LDA) e Within-Class Covariance Normalization (WCCN) per rimuovere possibili fastidi ed aumentare le probabilità che i dati ottenuti da canali di registrazione diversi possano essere correttamente assegnati ai relativi speaker.

In ultimo, l'esito dell'identificazione che è dato dal calcolo della cosine distance, cioè della somiglianza tra il vettore associato allo speaker nel database, ed il vettore ottenuto dal tentativo in questione.

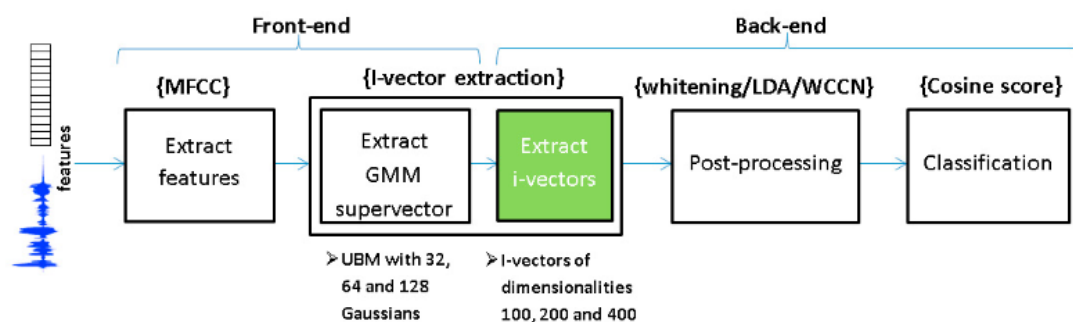


Figura 1-3: estrazione i-vector

In questo studio [1] sono state osservate diverse dimensioni per gli i-vector, ed è stato appurato che la versione a dimensione 400 è quella in grado di garantire performance migliori.

1.5 **Embedding fusion**

Abbiamo quindi visto i due vettori più utilizzati, protagonisti di questo lavoro di tesi. Come vedremo più avanti in questo elaborato, abbiamo avuto l'idea di fondere i due vettori, immaginando che potessero essere complementari, dato che concettualmente sono simili. Entrambi infatti convertono una registrazione vocale in un vettore di dimensione fissa che caratterizza lo speaker. Se da un lato gli x-vector fanno questo servendosi di una DNN, gli i-vector raggiungono l'obiettivo attraverso la JFA.

Se da un lato gli x-vector hanno performance di riconoscimento migliori specie su tracce brevi e sono lo stato dell'arte attuale, dall'altro gli i-vector sono in grado di godere di risultati migliori quando si tratta di riconoscere lo stesso speaker che registra da dispositivi diversi, proprio perché la JFA conferisce indipendenza dal canale di trasmissione, a discapito di un costo computazionale nettamente maggiore.

In letteratura non sembrano esserci progetti che documentano una simile fusione, obiettivo che ci poniamo in questo elaborato allo scopo di evidenziarne l'unicità e l'importanza.

Capitolo 2

RELATED WORK

Il progetto da cui partiremo è frutto del lavoro di tesi “Autenticazione Vocale con Reti Siamesi” [2] del mio collega Ciro Fusco, che ringrazio per il supporto offertomi.

Come è possibile intuire dal titolo, si tratta di un sistema di riconoscimento vocale basato sull'utilizzo di Reti Siamesi.

Innanzitutto esso si serve di un dataset di tracce audio in inglese Librispeech, per estrarre le feature necessarie e passarle in input alla rete.

L'estrazione è basata sugli x-vector, ed è realizzata grazie all'utilizzo del tool Kaldi.

Il cuore del progetto è la Rete Siamese, che si serve delle CNN per processare gli embeddings.

Ovviamente per creare un buon modello è indispensabile la fase di training e testing, realizzati attraverso appositi script in Python che tengono conto dei miglioramenti ottenuti durante le varie epoche di addestramento.

Per utilizzare il sistema sviluppato, è stata creata una web application raggiungibile tramite un indirizzo IP pubblico, caricata su un server Linux based, con CentOS come sistema operativo. Essa è stata creata con l'ausilio di Flask, ed il deploy è stato effettuato con Nginx, che è un reverse proxy.

In letteratura vi sono numerosi altri studi che analizzano il comportamento di sistemi di Speaker Recognition alternando l'implementazione degli i-vector a quella degli x-vector per effettuare dei confronti.

In tal senso uno degli studi più completi è questo condotto da Daniel Povey [3], da cui è emerso che in un sistema il cui training è effettuato su dataset come SRE, SWBD o SITW, la versione i-vector based ottiene risultati migliori del 12% rispetto agli x-vector, mostrando un tasso di errore minore ed una migliore DCF, cost detection function.

Successivamente le tracce audio vengono modificate con la tecnica della Data Augmentation, aggiungendo rumori di sottofondo, musica e riverbero. In questa casistica sembrano invece primeggiare gli x-vector con risultati migliori del 14%.

Allenando invece la rete sul celebre dataset Voxceleb, ancora una volta gli x-vector sembrano essere più precisi ed affidabili, con un tasso di errore migliore del 44% rispetto agli i-vector.

Sempre Povey in un altro studio [4] accenna in modo superficiale alla fusione di emeddings ed i-vector, affermando che un sistema allenato su NIST SRE10 la combinazione di vettori mostra risultati migliori su tracce di 10s, con un EER migliore del 28%, e DCF migliore del 11%.

Anche cambiando dataset con SRE16 gli embeddings fusion performano meglio dei vettori singoli, con un EER del +24% e +19% di DCF.

Un altro studio degno di nota è quello condotto l'anno scorso da Zhang e Bai [5], in cui gli autori forniscono una panoramica dell'intero campo di ricerca dello Speaker Recognition. Tale ambito infatti ha vari subtasks.

Alla base di tutti questi processi vi è sempre la fase di feature extraction, effettuabile tramite DNN/x-vector oppure UBM/i-vector.

Si parla di verification quando si vuole verificare se una frase è stata pronunciata da un ipotetico speaker, basandosi sulle sue registrazioni salvate in memoria. Gli algoritmi possono essere stage-wise, in cui nel front-end si estraggono informazioni e nel back-end si calcolano le somiglianze, oppure end-to-end quando si calcolano direttamente le somiglianze. L'identification è invece una versione one-to-many in cui si confronta la traccia audio in input con altre tracce di speaker diversi.

Speaker Diarization risolve invece la problematica di individuare correttamente tutti i vari speaker che stanno conversando all'interno di una singola traccia in input. Si parla invece di robust speaker recognition quando c'è bisogno di tecniche aggiuntive per aggirare problematiche di rumori di fondo o domain mismatch.

In generale il paper mira a sottolineare che l'avvento del deep learning ha portato lo speaker recognition ad un livello superiore dal punto di vista delle performance. Nonostante il successo evidente, non bisogna però perdere di vista la risoluzione di problematiche ancora aperte. Tra tutte il fatto che la fase di training richieda ancora enormi quantità di dati labellati e di risorse computazionali, e le funzioni di loss sono carenti di solide basi teoriche che permettano di poter valutare concretamente vantaggi e svantaggi dei vari modelli proposti.

Capitolo 3

MODIFICHE APPORTATE

In questo capitolo andremo ad illustrare le migliorie apportate al sistema appena descritto.

La fase di testing di quest'ultimo ha infatti evidenziato una problematica rilevante: il riconoscimento vocale basato sull'estrazione degli x-vector vede calare la propria efficacia dal 99% al 30% circa nel caso in cui si tenta l'accesso su un dispositivo diverso da quello utilizzato per la prima registrazione.

Si è quindi giunti alla conclusione che probabilmente la Rete Siamese resti in qualche modo troppo legata alle informazioni riguardanti il canale di trasmissione, contenute nell' x-vector.

In tal senso, è possibile che durante la prima registrazione venga memorizzata la traccia audio contenente la voce dell'utente a cui viene affibbiata anche l'informazione rappresentativa dell'hardware utilizzato per la registrazione.

Questo spiega infatti il motivo per il quale cambiando dispositivo, la rete sbaglia l'esito del riconoscimento nel caso positivo, cioè nel caso in cui si tratta della voce dello stesso utente che però sta registrando da un nuovo dispositivo.

Per cui erano stati delineati diversi possibili sviluppi futuri:

1. “Sporcare” tutte le tracce audio del dataset prima di avviare la fase di training della rete, tramite la tecnica della data augmentation.
2. Utilizzare un nuovo dataset per addestrare la rete, costituito da tracce audio di speaker registrati in simultanea con dispositivi diversi.
3. Sfruttare a pieno le funzionalità del tool open source Kaldi, estraendo oltre che x-vector anche gli i-vector.

In questo lavoro tesi abbiamo deciso di perseguire la terza opzione poiché si è rivelata essere la più veloce e sensata da realizzare.

La prima scelta avrebbe infatti richiesto un lavoro enorme sia quantitativo, in quanto si sarebbero dovute modificare tutte le tracce audio, che qualitativo poiché avremmo dovuto dapprima studiare la data augmentation e svariate altre tecniche di manipolazione di file audio, per poi saper applicarle in modo non banale.

La seconda possibilità era in realtà ottima, ma il più grande problema era rappresentato dal fatto che non esiste in letteratura un database con tali caratteristiche. Abbiamo quindi pensato di crearlo noi da zero predisponendo una line-up di appositi dispositivi nel laboratorio, ma ci siamo resi conto che sarebbe stato un compito molto difficile da assolvere, sia per tempo che per difficoltà oggettiva nel trovare una cinquantina di persone da poter portare fisicamente in laboratorio in un momento complicato come questo, data la pandemia.

Abbiamo quindi deciso di provare ad estrarre anche gli i-vector dalle tracce audio per poi combinarli con gli x-vector, confidando nella loro caratteristica di essere channel-independent.

3.1 Configurazione estrazione i-vector

Prima di tutto si è reso necessario configurare opportunamente i vari settaggi del tool Kaldi, per ottenere il corretto processo di estrazione degli i-vector.

Per fortuna gli ideatori del suddetto strumento hanno provveduto a stilare una guida che ci ha agevolato di molto tale processo. Noi quindi l'abbiamo seguita celermente.

Nel file `path.sh` abbiamo provveduto a modificare la variabile `Kaldi_Root`, assegnandole il percorso per raggiungere la cartella principale di Kaldi.

Seguendo il file `Readme` abbiamo poi creato il file `"enroll.sh"` che è il cuore pulsante di tutta la fase di estrazione. Esso infatti viene preceduto dal comando `bash` ed accompagnato da un file audio. Il file `sh` non fa altro che predisporre la corretta estrazione delle features lanciando script nativi che generano il vettore desiderato.

Nelle ultime righe non abbiamo fatto altro che facilitare la lettura del vettore risultante, trasformandolo da file `.ark` nativo, prima sottoforma di file di testo e successivamente come array di Numpy.

3.2 Estrazione combinata

Una volta sistemato lo script per l'estrazione degli i-vector, la strategia che si vuole perseguire consiste nell'estrarre sia gli i-vector che gli x-vector, combinandone l'estrazione in modo che per ogni traccia venga restituita la media pesata dei due vettori.

Lo script da modificare per realizzare ciò è quello denominato `fbanks_2d_extraction.py`.

```
1. def get_xvector(audio_file):  
  
2.     audio_dir = audio_file.rsplit('/',1)[0]  
3.     sub.call("bash /home/michele/Scrivania/progetto/Codici/ivector-xvector-  
         master/xvector/enroll_mod_ciro.sh "+audio_file+" 1",shell=True,  
         cwd=audio_dir)  
4.     xvector = np.load(audio_dir+"/data/embeddings.npy")  
5.     sub.call("rm -rf "+audio_dir+"/data",shell=True)  
  
6.     sub.call("bash /home/michele/Scrivania/progetto/Codici/ivector-xvector-  
         master/ivector/enroll_mod_mik.sh "+audio_file,shell=True, cwd=audio_dir)  
7.     ivector = np.load(audio_dir+"/data/i_vector.npy")  
8.     sub.call("rm -rf "+audio_dir+"/data",shell=True)  
  
9.     value = np.zeros((1,112))  
10.    ivector = np.append(ivector, value, axis=1)  
  
11.    return xvector * 0.4 + ivector * 0.6
```

Figura 3-1: fbanks_2d_extraction.py

Esso è infatti deputato all'estrazione dei vettori di ogni singola traccia presente nel dataset, i quali saranno poi effettivamente utilizzati per realizzare la fase di training della rete neurale.

Come è possibile osservare dal listato, abbiamo aggiunto le righe 6-7 per lanciare l'estrazione degli i-vector.

L'aggiunta della riga 11 ha attuato l'intenzione di voler ottenere in output una media pesata dei due vettori.

Apportate le seguenti modifiche, eravamo pronti a procedere con l'estrazione vera e propria sul dataset già utilizzato, LibriSpeech.

L'esecuzione dello script ha però manifestato un errore algebrico molto importante: non è possibile infatti sommare x-vector ed i-vector poiché si tratta di vettori aventi dimensioni diverse.

Analizzando dettagliatamente la documentazione di Kaldi abbiamo infatti scoperto che gli x-vector vengono rappresentati in vettori aventi dimensioni (1 , 512), mentre gli i-vector vengono estratti in una matrice avente 1 riga e 400 colonne, quindi (1 , 400).

La prima idea che abbiamo avuto è stata quella di modificare direttamente lo script `ivector-extraction.h`, che secondo la documentazione di Kaldi è il file principale che detta le modalità di estrazione degli i-vector.

Ed infatti tra i primi statements abbiamo notato che viene settata la dimensione degli i-vector a 400.

Sfortunatamente però settare tale parametro a 512 non ha sortito l'effetto desiderato.

A questo punto, per poter procedere col nostro esperimento bisognava trovare il modo di equalizzare la dimensione dei due vettori, che differivano di 112 colonne. Sono stati quindi identificati i seguenti correttivi:

1. Adattare gli x-vector alla forma degli i-vector, tagliando quindi le ultime 112 entry.
2. Adattare gli i-vector agli x-vector, aggiungendo un vettore di 112 colonne.

Sebbene la prima soluzione potesse sembrare più immediata e di facile applicazione, in realtà avrebbe avuto due controindicazioni pesanti: innanzitutto avrebbe falsato la media finale di ogni traccia audio, dato che rimuovere 112 colonne avrebbe significato grossomodo la perdita del 20% di informazioni.

Il side-effect principale riguardava il fatto che la rete utilizzata era configurata per le dimensioni native degli x-vector, (1 , 512). Portare gli xvector a (1 , 400) avrebbe quindi comportato modifiche lunghe e costose all'intera struttura della rete che era già stata utilizzata ed opportunamente configurata nella prima fase di questo esperimento.

L'unica strada percorribile era la seconda: aggiungere 112 colonne al vettore degli i-vector.

L'interrogativo giunto spontaneo ha riguardato il modo in cui riempire queste 112 colonne mancanti.

Anche qui le soluzioni potevano essere le più disparate:

- Inserire gli ultimi 112 elementi del vettore x-vector;
- ripetere alcune entry tra le 400 già presenti;
- inserire la media tra tutti i valori;
- inserire il valore più ricorrente;
- Inserire tutti 0;

L'ultima alternativa è quella che abbiamo deciso di adottare, per un motivo molto semplice: il discriminante è la valutazione che ognuna delle sopracitate scelte può avere sul calcolo della media ponderata dei due vettori.

Ognuna delle prime 4 scelte avrebbe comportato inevitabilmente un'alterazione più o meno consistente della media risultante.

L'ultima alternativa ha invece consentito di ottenere comunque una media abbastanza fedele della somma tra i due vettori.

Le prime 400 righe infatti risultano dalla media originaria di x-vector e ivector, mentre le ultime 112 righe rappresentano comunque un valore medio reale tra il l'i-esimo coefficiente i del xvector e lo 0 dell'ivector.

Ad onor del vero, abbiamo sperimentato anche la prima alternativa, cioè quella di ricopiare le ultime 112 righe dell'xvector.

Per cui avevamo 400 righe di media effettiva tra i due vettori, mentre le altre 112 erano dominate dal valore di xvector. Quella che ci sembrava essere la soluzione ideale, si è rivelata essere poco efficace, perché così facendo gli xvector avrebbero avuto un peso molto maggiore nel calcolo della media. Ed infatti questo è quello che si è verificato, riconducendoci

al problema di partenza dell'accesso multi-device che abbiamo già sviscerato in precedenza.

Da qui la necessità di introdurre le istruzioni di riga 9-10. La riga 11 non fa altro che esprimere la volontà di ottenere la media pesata dei due vettori, dando il 60% di peso agli ivector proprio per quanto detto pocanzi.

3.3 Estrazione multiprocessing

Una volta ultimato lo script di estrazione, eravamo finalmente pronti a lanciarlo per ottenere i vettori desiderati sull'intero dataset.

Dopo 24 ore di esecuzione ci siamo accorti con enorme stupore che la percentuale di avanzamento era veramente molto bassa. Questo ci ha portato a stimare una durata dell'intero processo di estrazione pari circa a 10 giorni.

Ci siamo quindi trovati di fronte ad un tempo ragionevolmente troppo lungo da sopportare al fine di poter proseguire col nostro esperimento, anche in virtù del fatto che probabilmente avremmo avuto l'esigenza di dover far ripartire l'estrazione in seguito alla correzione di errori vari o sotto condizioni differenti.

Siamo pervenuti alla consapevolezza di dover parallelizzare in qualche modo l'esecuzione del nostro script. Quindi lo abbiamo riprogettato con l'ausilio della preziosa libreria “multiprocessing” di

Python la quale, una volta studiata, si è rivelata essere di fondamentale importanza per raggiungere il nostro scopo.

Figura 3-2:fbanksmulti_extraction.py

```
1. import os
2. import re
3. from io import StringIO
4. from pathlib import Path
5. import numpy as np
6. import pandas as pd
7. import librosa
8. import subprocess as sub
9. import multiprocessing as mp
10. import python_speech_features as psf
11. BASE_PATH =
    '/home/fusco/Speaker_Recognition/feature_extraction/LibriSpeech'
12. OUTPUT_PATH = 'fbanks'
13. np.random.seed(42)

14. def read_metadata():
15.     with open(BASE_PATH + '/SPEAKERS.TXT', 'r') as meta:
16.         data = meta.readlines()
17.         data = data[11:]
18.         data = "".join(data)
19.         data = data[1:]
20.         data = re.sub(' +|', '', data)
21.         data = StringIO(data)
```

```

22. speakers = pd.read_csv(data, sep='|', error_bad_lines=False)
23. speakers_filtered1 = speakers[(speakers['SUBSET'] == 'train-clean-100')]
24. speakers_filtered = speakers_filtered1.copy()
25. speakers_filtered['LABEL'] =
    speakers_filtered['ID'].astype('category').cat.codes
26. return speakers_filtered

27. def get_xvector(audio_file):
28.     audio_dir = audio_file.rsplit('/', 1)[0]
29.     sub.call(
30.         "bash /home/fusco/webapp/ivector-xvector-
        master/xvector/enroll_mod_ciro.sh " + audio_file + " 1",
31.         shell=True, cwd=audio_dir)
32.     xvector = np.load(audio_dir + "/data_x/embeddings.npy")
33.     sub.call("rm -rf " + audio_dir + "/data_x", shell=True)
34.     sub.call(
35.         "bash /home/fusco/webapp/ivector-xvector-
        master/ivector/enroll_mod_mik.sh " + audio_file,
36.         shell=True, cwd=audio_dir)
37.     ivector = np.load(audio_dir + "/data/i_vector.npy")
38.     sub.call("rm -rf " + audio_dir + "/data", shell=True)

39.     value = np.zeros((1,112))
40.     ivector = np.append(ivector, value, axis=1)

41.     return xvector * 0.4 + ivector * 0.6

```



```

42. def assert_out_dir_exists(index):
43.     dir_ = OUTPUT_PATH + '/' + str(index)
44.     if not os.path.exists(dir_):
45.         os.makedirs(dir_)
46.         print('crated dir {}'.format(dir_))
47.     else:
48.         print('dir {} already exists'.format(dir_))
49.     return dir_

50. def estrai(id):
51.     dir_ = BASE_PATH + '/train-clean-100/' + str(id) + '/'
52.     index_target_dir = assert_out_dir_exists(id)
53.     files_iter = Path(dir_).glob('**/*.flac')
54.     files_ = [str(f) for f in files_iter]
55.     print('done for id: {}'.format(id))
56.
57.     for i, f in enumerate(files_):
58.         fbanks = get_xvector(f)
59.         np.save(index_target_dir + '/' + str(i) + '.npy', fbanks)
60.
61.     print('done for id: {}'.format(id))
62.     print("")

```

```

63. def main():
64.     speakers = read_metadata()
65.     print('read metadata from file, number of rows in in are:
        {}'.format(speakers.shape))
66.     print('numer of unique labels in the dataset is:
        {}'.format(speakers['LABEL'].unique().shape))
67.     print('max label in the dataset is: {}'.format(speakers['LABEL'].max()))
68.     print('number of unique index: {}, max index:
        {}'.format(speakers.index.shape, max(speakers.index)))
69.     lista_id = np.array([0])
70.
71.     #CREA LA LISTA CON TUTTI GLI ID
72.     for index, row in speakers.iterrows():
73.         subset = row['SUBSET']
74.         id_ = row['ID']
75.         lista_id = np.append(lista_id, id_)
76.
77.     lista_id = np.delete(lista_id, 0)
78.     print(lista_id)
79.
80.     # CREA IL POOL
81.     p = mp.Pool(mp.cpu_count() - 1)
82.     async_result = p.map_async(estrai, lista_id)
83.     p.close()
84.     p.join()
85.     print('All done, YAY!, look at the files')

86.     main()

```

Come possiamo osservare il metodo `get_xvector` resta totalmente invariato, così come `read_metadata` che recupera le informazioni relative agli speaker del dataset, ed il metodo `assert_out_dir_exists` che verifica l'esistenza della directory esaminata.

Il `main()` conserva la prima fase di raccolta delle informazioni relative agli speaker, ma la prima aggiunta rilevante riguarda le righe 72-75.

Qui costruiamo la lista di tutti gli id, che sono utili perché rappresentano mediante un numero l'intera directory di un determinato speaker.

Il motore dell'esecuzione parallela è la riga 81 in cui andiamo ad istanziare la classe `Pool` della libreria di `multiprocessing`.

Tale classe si occupa infatti della gestione di un vero e proprio pool di “workers”, cioè di `n` processi da schedulare in parallelo, ognuno in modo indipendente con la propria coda di task.

Il numero di workers viene passato come parametro con l'istruzione `mp.cpu_count - 1`. Siccome abbiamo a disposizione un server performante, abbiamo quindi potuto contare su una trentina di processi, lasciandone solamente uno libero per permettere al server di assolvere ai task di gestione interna.

Il metodo `map_async()` di riga 82 non fa altro che lanciare l'esecuzione della funzione “estrai”, che viene applicata su un elemento iterabile, in questo caso la lista degli id.

Così facendo tale input viene mappato su tutti i processori disponibili, e l'output viene via via raccolto dai vari processori.

Terminato un task, ogni processore comunica l'output e passa al prossimo job nella propria coda.

In questo caso ogni processore in modo parallelo ed indipendente esegue la funzione estrai su un id della lista.

La funzione estrai() prende infatti in input un id, ed una volta verificata l'esistenza della cartella relativa allo speaker identificato, viene chiamata l'estrazione combinata degli x-vector ed i-vector per ogni traccia audio presente nella suddetta directory.

In altre parole ogni processore del pool effettua un'estrazione combinata dei vettori su tutte le tracce audio di uno speaker, cioè di una cartella contrassegnata da uno degli id della lista generale.

Infine, abbiamo il metodo join() permette di bloccare l'esecuzione del programma principale fino a quando tutti i processi coinvolti non hanno terminato la propria esecuzione.

3.4 Webapp

Per ciò che concerne le fasi di training e testing della rete siamese, non si è reso necessario apportare modifiche.

Viceversa, profondi cambiamenti sono stati approntati a tutta la sezione del progetto relativa alla “Webapp”, sia dal punto di vista logico che da quello meramente grafico.

Per maggiore chiarezza, partiamo dal primo aspetto.

Figura 3-3: webapp.py

```
1. @app.route('/login/<string:username>', methods=['POST'])
2. def login(username):
3.     exists = _check_name(request, username)
4.     if not exists:
5.         return Response('USER_NOT_EXISTS', mimetype='application/json')
6.     filename = _save_file(request, username)
7.     vector_fusion = extract_fusion(filename)
8.     embeddings_fusion = get_embeddings_fusion(vector_fusion)
9.     stored_embeddings_fusion = np.load(DATA_DIR + username +
    '/embeddings_fusion.npy')
10.    stored_embeddings_fusion = stored_embeddings_fusion.reshape((1, -1))
11.    distances_embed_fusion = get_cosine_distance(embeddings_fusion,
    stored_embeddings_fusion)
12.    print('mean distances embeddings_fusion',
    np.mean(distances_embed_fusion), flush=True)
13.    now = date.now()
14.    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
```

```

15. with open(DATA_DIR + username + '/access.log','a') as log:
16.     log.write("\nTentativo di accesso in data: " + dt_string+ "\n\n")
17.     log.write('\n\nmean distances vector_fusion embeddings ')
18.     log.write(str(np.mean(distances_embed_fusion)))
19.     if distances_embed_fusion <= .35:
20.         log.write("\n\nRisultato Media Pesata: POSITIVO")
21.     else:
22.         log.write("\n\nRisultato Media Pesata: NEGATIVO")
23.     log.write("\n\n\n=====FINE=====\\n\\n\\n")
24.     data_obj = {"vector_fusion embeddings": str(np.mean ( distances
        _embed_fusion))} data = json.dumps(data_obj)
25.     if distances_embed_fusion <= .35:
26.         return Response(data, 250, mimetype='application/json')
27.     else:
28.         return Response(data,status='FAILURE', mimetype='application/json')

```

Le uniche modifiche apportate alla fase di riconoscimento, alias login riguardano semplicemente l'applicazione delle nuove metriche contraddistinte dall'aggettivo "fusion".

Resta quindi invariata la logica di base, cioè il confronto tra la traccia inviata alla prima registrazione (caricata in "stored_embeddings") e la traccia registrata durante il nuovo tentativo di accesso (su cui avviene l'estrazione combinata dei due vettori, "vector_fusion"), computato tramite il metodo `get_cosine_distance()`.

Lo stesso discorso vale anche per la fase di registrazione, in cui cambia solo il vettore salvato nella cartella, fusion invece di xvector.

```

29. @app.route('/check/<string:username>', methods=['POST', 'GET'])
30. def check(username):
31.     dir_ = DATA_DIR + username
32.     if os.path.exists(dir_):
33.         print("User esiste")
34.         return Response('USER_ALREADY_EXISTS',
35.                         mimetype='application/json')
35.     else:
36.         return Response('USER_NOT_EXISTS', mimetype='application/json')

37. @app.route('/register/<string:username>', methods=['POST', 'GET'])
38. def register(username):
39.     exists = _check_name(request, username)
40.     if not exists:
41.         filename = _save_file(request, username)
42.         vector_fusion = extract_fusion(filename)
43.         np.save(DATA_DIR + username + '/vector_fusion.npy', vector_fusion)
44.         embeddings_fusion = get_embeddings_fusion(vector_fusion)
45.         mean_embeddings_fusion = np.mean(embeddings_fusion, axis=0)
46.         np.save(DATA_DIR + username + '/embeddings_fusion.npy',
47.                 mean_embeddings_fusion)
47.         return Response('SUCCESS', mimetype='application/json')
48.     else: return Response('USER_ALREADY_EXISTS')

```

E' stata poi introdotta la funzione di check che controlla se il nome utente (e quindi la relativa cartella) esiste già all'interno del file system al fine di migliorare l'esperienza grafica, che illustreremo a breve.


3.5 Front-end ed utilizzo

Notevoli cambiamenti sono stati apportati nell'ambito UX/UI, sia per rendere la webapp più accattivante agli occhi dell'utente, ma anche per migliorarne l'esperienza e la facilità di utilizzo.

Infatti come è possibile osservare dalle prossime immagini abbiamo conservato l'idea e l'organizzazione principale, riprogettandone il layout e lo stile in un template responsive godibile da qualsiasi tipo di dispositivo.

Scegli cosa fare Riconosci ▾

Inserisci il nome associato alla voce che vuoi riconoscere

Inserisci il nome 

Leggimi

L'anno 1913 mi diede un momento d'esitazione. Mancava il tredicesimo mese per accordarlo con l'anno. Ma non si creda che occorran tanti accordi in una data per dare rilievo ad un'ultima sigaretta. Molte date che trovo notate su libri o quadri preferiti, spiccano per la loro deformità. Per esempio il terzo giorno del secondo mese del 1905 ore

Registra **Stop**

Timer

Registrazioni

Invia

Leggimi

Tutti i file registrati saranno salvati ed utilizzati ai soli scopi di cui gli utenti

Figura 3-4: home page originaria

Ecco invece il nuovo layout.

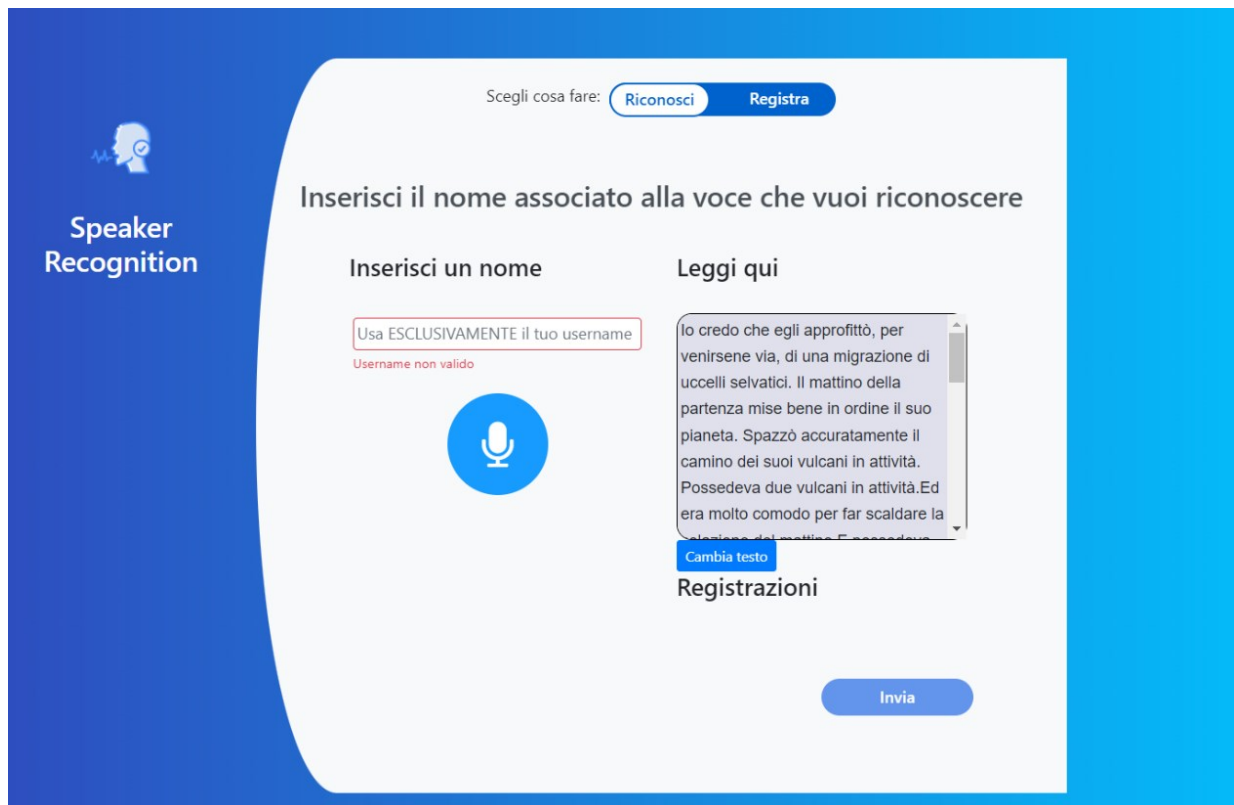
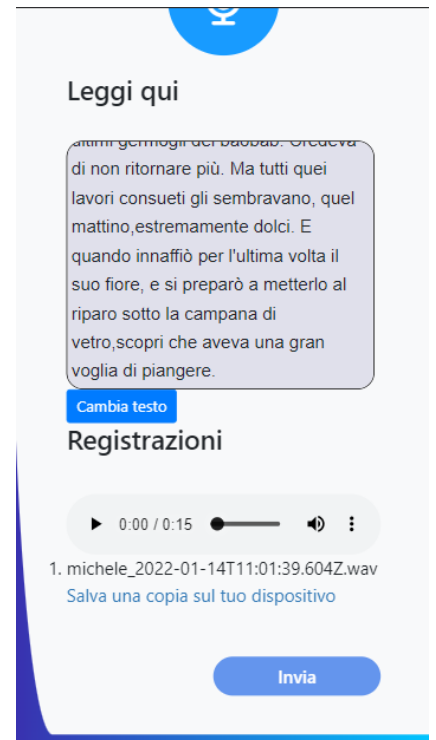
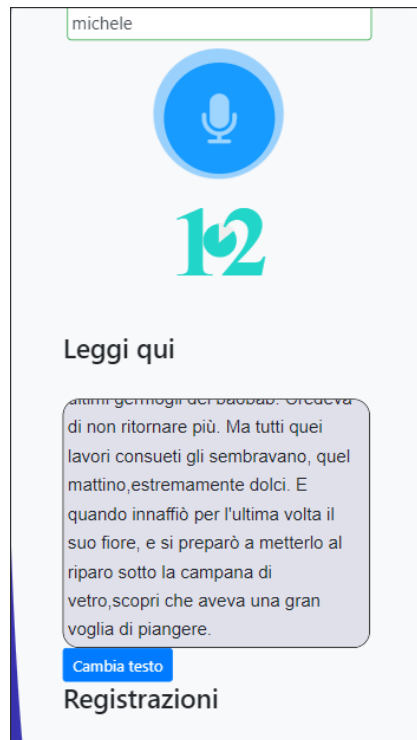


Figura 3-5, 3-6: versione desktop e mobile



In entrambi i formati la prima cosa che viene chiesta all'utente è l'azione da compiere, tra Riconosci oppure Registra.

Una volta selezionato il toggle, l'utente deve digitare il proprio username, associato alla voce.

Oltre a migliorare lo stile di questo input tag, abbiamo introdotto un controllo a runtime sul nome digitato. Per ogni lettera digitata infatti, interroghiamo il server tramite richieste AJAX per sapere se quel nome esiste già, sfruttando la routine “check” che abbiamo introdotto nel paragrafo immediatamente precedente.

Nel caso “Registra” presentiamo un messaggio di errore visibile nella figura 3.5 se il nome scritto è già stato utilizzato da qualcun altro.

Se invece il nome non risulta già abbinato ad una voce, coloriamo il contorno di verde, come in figura 3.6.

Così facendo agevoliamo l'utente che non deve procedere per tentativi nella creazione dell'username, evitando la situazione tediosa di dover ripetere la registrazione vocale se questa era stata fatta con un nome non utilizzabile.

Lo stesso ragionamento vale per il caso “Riconosci”, invertendo semplicemente i criteri di scelta sull'esistenza del nome.

Digitato il nome utente, basta premere sul bottone del microfono per registrare la propria voce. Una volta fornito il consenso, inizia un countdown animato e di durata preimpostata(a seconda dell'azione scelta) per notificare che la registrazione è in atto, concetto evidenziato anche dall'animazione di “pulsing” del microfono.

Terminato l'intervallo di recording, scompare il timer ed appare la nota vocale (visibile nell'ultima immagine), in modo tale che l'utente può riascoltarla per controllarne la qualità o per conservarne una copia.

Una volta inviata al server, esso impiega circa 25-30 secondi per dare una risposta relativa all'esito del riconoscimento.

Se si tratta di un tentativo di accesso, viene semplicemente mostrato un pop-up con la risposta. Se invece si tratta della prima registrazione l'utente viene accolto con un'apposita pagina di benvenuto in cui viene notificato il buon esito dell'operazione.

Anche questa pagina è stata riprogettata, seguendo lo stesso stile conferito all'homepage.

Figura 3-6: Pagina di notifica, PRIMA e DOPO

**Ciao Ciro. Sei stato
correttamente registrato**

Torna alla Home



3.6 Back-end

Per ciò che concerne il codice che si trova alla base della nostra webapp, tralasciamo la disamina delle pagine HTML e del foglio di stile CSS per non annoiare il lettore.

Lo stylesheet infatti non offre particolari spunti di riflessione, trattandosi di un'elencazione di settaggi grafici.

Le pagine HTML hanno invece conservato la struttura generale originaria, arricchite delle librerie e degli attributi di classe di Bootstrap necessari alla riprogettazione del layout.

Ci concentriamo invece sulle novità introdotte nel file JavaScript, che presentiamo di seguito.

```
var elems = document.getElementsByTagName('li');
var actionSelected = "login";

function assignActionValue(action) {
    actionSelected = action;

    if( actionSelected == "login"){
        document.getElementById("action-label").innerHTML
= "Inserisci il nome associato alla voce che vuoi
riconoscere";
        document.getElementById("email-
validation").innerHTML = "Username non valido";
    }else{
        document.getElementById("action-label").innerHTML
= "Inserisci il nome che vuoi associare alla voce";
        document.getElementById("email-
validation").innerHTML = "Utente già esistente";
    }

    var actionSelect = actionSelected;
    actionSelect.onchange = loadWikiArticle;
}
```

```
function getActionSelected() {
    var action;
    Array.from(elems).forEach(v =>
v.addEventListener('click', function() {
    action = this.getAttribute("value");
    //console.log("Ho passato: ",action);
    assignActionValue(action);
    controlla(); //check username
}));
}
```

Questo primo blocco di codice è stato introdotto per supportare il toggle grafico della scelta tra Riconosci e Registra.

La seconda funzione `getActionSelected()` prende i due elementi e vi setta un Listener per l'evento di click. Quindi ogni volta che l'utente preme sulla scelta, viene chiamata la funzione `assignActionValue()` che setta il valore dell'azione desiderata.

```
// Funzione di controllo sulla durata minima delle
registrazioni
function timer(s){

    getActionSelected();
    var action = actionSelected;
    ++seconds;
    if(action === "register" && seconds > 30) {

        stopRec();
    }else{
        if (action === "login" && seconds > 15){
            /*stopButton.disabled = false;
            element.style.background = "green";
document.getElementById("action").disabled=false;*/
            stopRec();
        }
    }
}
```

La funzione di timer ha invece lo scopo di settare una durata preimpostata delle registrazioni effettuate dagli utenti, per due motivi:

Per prima cosa evitiamo che l'utente produca audio troppo lunghi o troppo corti che potrebbero inficiare sull'esito del riconoscimento, ma in secondo luogo evitiamo anche una problematica poco frequente riscontrata sul browser Safari, in cui abbiamo osservato un time stretching degli audio. Essi infatti, durante l'elaborazione, venivano allungati di qualche secondo rispetto alla loro durata originale, provocando ripercussioni sia sulla qualità dell'audio che sulla capacità di riconoscimento.

```
//Listener onkeyup
function controlla(){

    var username =
document.getElementById('username').value;
    var usernameId = document.getElementById('username');
    //getActionSelected();
    var action = actionSelected;

    if(username.length == 0){
        usernameId.classList.add('is-invalid');
        usernameId.classList.remove('is-valid');

        return false;
    }else{
        var xhr = new XMLHttpRequest();
        xhr.onloadend = function (e) {

            if (this.readyState === 4) {
                console.log("Server returned: ",
e.target.responseText);
                var response = e.target.responseText;
                if (action === 'register') {
                    if (response == 'USER_ALREADY_EXISTS'){
                        usernameId.classList.add('is-invalid');
                        usernameId.classList.remove('is-valid');
                    }
                }
            }
        }
    }
}
```

```

}else{
    usernameId.classList.remove('is-invalid');
    usernameId.classList.add('is-valid');
}
} else if (action === 'login') {
    if (response === 'USER_ALREADY_EXISTS'){
        usernameId.classList.remove('is-invalid');
        usernameId.classList.add('is-valid');
    }else{
        usernameId.classList.add('is-invalid');
        usernameId.classList.remove('is-valid');
    }
}
}
};
var fd = new FormData();
fd.append("username", username);
xhr.open("POST", "/" + "check" + "/" + username,
true);
xhr.send(fd);
}
}

```

Questa funzione realizza invece il controllo a runtime sul nome utente che abbiamo illustrato nel paragrafo antecedente.

Se la lunghezza del nome è 0 si dà il messaggio di errore in rosso, altrimenti si crea la richiesta in AJAX e si invia il nome da esaminare al server tramite una richiesta POST ed invocando la procedura check.

A seconda della risposta restituita dal server, USER_ALREADY_EXISTS oppure USER_NOT_EXISTS, e in base al tipo di azione selezionata, si settano i giusti attributi di classe per visualizzare il feedback grafico.

L'ultima funzione aggiunta è drawTimer(), usata per disegnare il timer animato sfruttando la potente libreria d3.js per grafici e animazioni.

Capitolo 4

TESTING E SPERIMENTAZIONE

Ogni software che si rispetti ha bisogno di un'accurata fase di analisi e testing allo scopo di osservarne il reale comportamento e di conseguenza valutarne l'efficacia.

Pertanto anche noi abbiamo condotto una sperimentazione in tal senso sulla base di una decina di utenti. Siamo ben consapevoli che per avere dati statistici più affidabili dovremmo coinvolgere minimo 50 persone, ma per ovvie ragioni questo non è stato possibile e quindi presentiamo i seguenti dati come una fotografia approssimativa del comportamento atteso.

La modalità con cui abbiamo svolto i nostri test è la seguente: ogni utente si è registrato sulla piattaforma con due dispositivi, uno smartphone ed un pc.

Per ogni dispositivo, in seguito alla registrazione, sono stati effettuati:

1. 5 tentativi di accesso sul dispositivo da cui ci si è registrati;
2. 5 tentativi di accesso all'account registrato con l'altro dispositivo (quindi da pc l'utente cerca di farsi riconoscere come utente-cell e viceversa)
3. 5 tentativi di accesso agli account di tutti gli altri.

Per quanto riguarda il primo caso, abbiamo avuto conferma che il sistema funziona perfettamente come nel progetto da cui siamo partiti.

Su 100 tentativi di accesso al proprio dispositivo di registrazione il sistema ha confermato la stessa percentuale di efficacia riscontrata nel lavoro svolto in precedenza, ossia il 98%.

Concentriamoci ora sulla seconda casistica: l'accesso dello stesso utente da dispositivi diversi. Nella versione del progetto con soli x-vector, avevamo evidenziato che l'efficacia calava al 30% circa. In questa nuova versione, con la fusione degli i-vector, possiamo osservare un oggettivo miglioramento: su 70 tentativi totali il sistema ha sbagliato solo 13 volte, con un tasso di efficacia dell'81%.

Per ciò che riguarda il terzo tentativo, ossia il fatto che ognuno dovesse provare ad accedere 5 volte a tutti gli altri account, abbiamo deciso di realizzare questo test creando uno script in python, dato l'elevato numero di combinazioni generate.

Abbiamo quindi generato un totale di 900 tentativi di accesso, sui quali il sistema ha mostrato di capire correttamente nel 89% dei casi, che chi stava tentando di accedere non era effettivamente il proprietario. La soglia del riconoscimento è attualmente impostata a 0.20, ma abbassandola a 0.19 il sistema guadagna un punto percentuale di efficacia, toccando quindi il 90%.

CONCLUSIONI

Come è possibile osservare dai risultati mostrati nel capitolo precedente, possiamo affermare che il sistema ha sicuramente giovato dell'introduzione degli i-vector.

Essi infatti non hanno minimamente minato l'accuratezza del riconoscimento sul dispositivo di registrazione, che era la peculiarità osservata nella versione precedente x-vector based.

Inoltre la loro fusione sembra aver effettivamente migliorato le prestazioni nel caso di accesso multi-dispositivo, passando dal 30% all'81% di precisione nel riconoscimento.

Possiamo quindi evidenziare che l'indipendenza dal canale degli i-vector si è rivelata essere un fattore chiave ed imprescindibile per casi d'uso del genere.

Infine essi hanno migliorato anche la capacità del sistema di riconoscere falsi positivi che tentano di accedere ad un account nel 90% dei casi.

Ovviamente ribadiamo innanzitutto che c'è bisogno di aumentare il numero di utenti da testare, per vedere se le nostre statistiche vengono confermate anche su numeri più grandi.

Detto questo, se da un lato abbiamo osservato risultati incoraggianti rispetto alla versione precedente, dall'altro non possiamo ancora essere soddisfatti delle performance e della precisione, specie se valutassimo

l'introduzione di questo sistema in ambienti critici in cui anche un solo accesso concesso ad una persona non autorizzata, potrebbe essere fatale.

Il sistema può essere quindi ulteriormente migliorabile.

Si potrebbe ad esempio aumentare il numero degli speaker nel dataset di training, facendo in modo che la rete impari a distinguere sempre più voci.

O ancora si potrebbero introdurre tracce audio contenenti lievi rumori di sottofondo, per rendere il sistema più robusto anche in questa casistica.

Lato webapp invece si potrebbe valutare l'utilizzo e l'impatto di una libreria js di registrazione differente, che magari garantisca una migliore qualità dell'audio registrato.

Ma lo sviluppo futuro che crediamo possa essere decisivo consiste nel cambiare dataset di addestramento, ricreandolo da zero.

Quello attuale, Librispeech, è interamente registrato in mp3 e poi convertito in flac per esigenze di memoria.

Nella nostra webapp noi invece registriamo audio in formato .wav.

Crediamo quindi che la differenza di formato tra le tracce audio che la rete ha usato per addestrarsi e quelle che le vengono passate dalla webapp per il riconoscimento, possa essere cruciale e possa impattare con un certo peso sulle prestazioni.

Suggeriamo perciò di creare un dataset usando il nostro tool di registrazione da più dispositivi, in modo tale da uniformare il formato di registrazione e migliorare il riconoscimento di uno stesso speaker da canali di registrazione differenti.

BIBLIOGRAFIA

- [1] Noor Salwani Ibrahim, Dzati Athiar Ramli, 2018, *I-vector Extraction for Speaker Recognition Based on Dimensionality Reduction*
<https://www.sciencedirect.com/science/article/pii/S1877050918314042>
- [2] Fusco Ciro, 2021, *Autenticazione Vocale con Reti Siamesi*
- [3] David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, Sanjeev Khudanpur , 2018, *X-VECTORS: ROBUST DNN EMBEDDINGS FOR SPEAKER RECOGNITION*
https://www.danielpovey.com/files/2018_icassp_xvectors.pdf
- [4] David Snyder, Daniel Garcia-Romero, Daniel Povey, Sanjeev Khudanpur , 2017, *Deep Neural Network Embeddings for Text-Independent Speaker Verification.*
http://danielpovey.com/files/2017_interspeech_embeddings.pdf
- [5] Zhongxin Bai, Xiao-Lei Zhang , 2021, *Speaker Recognition Based on Deep Learning: An Overview*
<https://doi.org/10.1016/j.neunet.2021.03.004>