Audio DeepFake Detection

Riccardo Baratin, Andrea Di Ubaldo

May 2023

1 Introduzione

L'audio Deepfake è una forma di manipolazione dell'audio in cui vengono utilizzati tecniche di Machine Learning e Deep Learning, al fine di generare audio sintetico che sembra essere prodotto da una persona o da una fonte audio, ma che in realtà è stato creato da una macchina. L'obiettivo principale dell'audio Deepfake è quello di falsificare l'audio in modo da ottenerne un altro che sia autentico e convincente, ma che in realtà è stato generato artificialmente. Questa tecnologia può essere utilizzata per scopi ludici, ma purtroppo viene impiegata per diffondere disinformazione o causare reati, come la frode o la truffa.
Gli attacchi si possono suddividere in tre tipologie principali:

- Replay-based: interventi malevoli incentrati a riprodurre una registrazione vocale originale con lo scopo di passare per l'individuo che parla nella registrazione.
- Imitation-based: attacchi che hanno lo scopo di imitare la voce di un individuo per fargli pronunciare parole che lui non ha mai detto prima, ma che sono state dette da un'altra persona in una registrazione differente.
- Synthetic-based: questa categoria di attacco converte un testo in segnale audio istantaneamente, nel modo più accettabile e irriconoscibile possibile da una persona o da un dispositivo di riconoscimento automatico. Il testo in input viene convertito da un modello o da un metodo di feature extraction in una rappresentazione intermedia di piccola dimensione, spesso in caratteristiche linguistiche o spettrogrammi di Mel. In seguito un modello aggiuntivo, detto vocoder, converte questa rappresentazione in formato audio. L'applicazione del Deep Learning nello sviluppo di nuovi vocoder costituisce un'ampia area di ricerca in questo settore.

Dati gli sviluppi tecnologici nell'ambito del DeepFake sia dalla qualità che dalla facile reperibilità degli strumenti in grado di produrre materiale sintetico di discreta qualità, la ricerca ha iniziato a studiare modelli di DeepFake detection. Il modello scelto per sviluppare il progetto è SpecRNet [3], una DNN capace di distinguere segnali audio originali da quelli sintetici in modo efficace ed efficiente, visto che richiede il 40% in meno del tempo necessario per analizzare un

campione audio, pur fornendo prestazioni comparabili ad un'architettura LCNN – uno dei modelli migliori nella DeepFake Detection.
L'obiettivo di questo progetto è quello di implementare un'architettura simile a quella di SpecRNet utilizzando il framework Keras, allenare e testare un modello su un dataset più completo confrontando la tecnica di estrazione delle feature da loro utilizzata, con quella basata sugli spettrogrammi di Mel.

2 MFCC

Il modello in questione è stato allenato su un set di registrazioni audio originali e un set di registrazioni sintetiche generate utilizzando gli audio originali e 8 differenti modelli DNN che generano attacchi TTS. SpecRNet opera sugli spettrogrammi bi-dimensionali dei segnali audio, invece di allenare il modello direttamente con i file audio, dato che lavori recenti hanno notato incrementi di prestazioni sulle prestazioni allenate con questa forma di dato. I segnali audio udibili dall'orecchio umano sono spesso composti da un insieme di frequenze differenti che rendono questi suoni difficili da analizzare anche per una rete neurale. Attraverso lo spettro di un segnale sonoro è possibile rappresentare tutte le frequenze – utilizzando come unità di misura gli Hertz(Hz), presenti in una registrazione in un dato momento, con la relativa ampiezza, ovvero la potenza del segnale. Uno spettrogramma di un segnale illustra lo spettro al variare del tempo, mostrando le frequenze e l'ampiezza di ciascuna con un colore più scuro se il segnale è di bassa intensità o, al contrario, più acceso.
Gli spettrogrammi vengono calcolati applicando la trasformazione di Fourier discreta, più costosa, o la trasformazione veloce che mostra come il segnale varia nel tempo nella sua interezza, pertanto non permette di notare le variazioni di ciascuna frequenza nel tempo. In alternativa si può ricorrere alla trasformazione di Fourier a breve termine, che divide l'audio in sezioni più piccole, applica la trasformazione veloce su ciascuna parte e combina ciascun risultato per ricostruire come ogni frequenza varia nel tempo.
Gli spettrogrammi vengono successivamente lavorati applicando ulteriori passi con l'obiettivo di estrarre le feature spettrali dalle frequenze in scala lineare presenti negli audio, chiamati LFCC. L'obiettivo di questo progetto è quello di implementare la stessa architettura di SpecRNet utilizzando il framework Keras, allenare e testare un modello su un dataset più ricco, confrontando i risultati prodotti usando gli LFCC con quelli ottenuti da un'altra tecnica di estrazione delle feature, gli spettrogrammi di Mel (MFCC). Questo tipo di spettrogramma mostra l'andamento delle frequenze di ciascun segnale in scala Mel, ovvero logaritmica, e le relative ampiezze in Decibel(DB), un unità di misura che cresce all'aumentare esponenziale della potenza. Il silenzio in Decibel assume valore 0, mentre un segnale 10 volte più potente ha valore 10, un segnale 100 volte più potente ha valore 20, e così via. L'impiego delle frequenze in scala Mel consente agli esseri umani di discriminare meglio le diverse frequenze presenti in una registrazione rispetto all'utilizzo della scala lineare [1]. Applicando gli stessi passaggi visti per creare gli LFCC a partire dagli spettrogrammi, è possibile estrarre gli

MFCC dagli spettrogrammi di Mel. Sebbene gli LFCC mostrano le caratteristiche degli spettrogrammi a maggiore risoluzione nelle alte frequenze rispetto agli MFCC, questi ultimi riducono la dimensionalità dell'audio compattando le informazioni più importanti delle registrazioni e diminuiscono la sensibilità al rumore di fondo, concentrandosi sulle informazioni spettrali più importanti. Pertanto si vuole confrontare le due tecnica di estrazione delle feature audio per verificare che gli MFCC siano sufficienti per il task in questione.

3 Dataset

3.1 Tecniche di generazione audio sintetici

I primi vocoder per gli attacchi TTS (Text-to-Speech) si basavano su modelli HMM (Hidden Markov Models) per generare voci sintetiche a partire da coppie di sequenze di parametri acustici e audio corrispondenti. Questi modelli apprendevano le distribuzioni probabilistiche dei parametri acustici rispetto all'audio, consentendo di generare nuove sequenze di parametri acustici basate sul testo da sintetizzare. Successivamente, questi parametri venivano utilizzati per sintetizzare la voce secondo un modello vocale specifico.
In seguito vennero sviluppati modelli generativi auto-regressivi, come WaveNet, che hanno rivoluzionato l'approccio alla sintesi vocale. Questi modelli si basano su reti neurali ricorrenti e trasformer e sono particolarmente adatti a generare forme d'onda audio realistiche. Utilizzano le sequenze di dati acustici dei campioni precedenti per generare quelli successivi, consentendo una sintesi vocale di alta qualità. Tuttavia, questi modelli sono computazionalmente molto costosi.
Per mitigare il costo computazionale, sono stati introdotti approcci basati su reti neurali convoluzionali flow-based, come Parallel WaveNet. Questi modelli cercano di modellare distribuzioni di probabilità complesse in distribuzioni più semplici, semplificando il processo di generazione delle forme d'onda audio. Ciò consente di ottenere risultati di sintesi vocale simili a quelli dei modelli autoregressivi, ma con un costo computazionale inferiore.
Un'altra tecnica di generazione di audio sintetici nel contesto dei vocoder è Direct Maximum Likelihood. Quest'ultima cerca di trovare i parametri del modello che massimizzano la probabilità dei dati osservati. Un esempio di modello di rete neurale che sfrutta questa tecnica è WaveGlow, il quale genera nuove forme d'onda audio a partire dagli audio del training set che sono probabilisticamente più simili al risultato da ottenere.
Tra i modelli di generazione di audio sintetici più recenti vi è sicuramente il Generative Adversarial Network (GAN). Quest'ultimo ha permesso di creare vocoder che sfruttano la generazione realistica GAN per sintetizzare forme d'onda audio di alta fedeltà e qualità. L'obiettivo principale di un vocoder basato su GAN è generare forme d'onda audio che siano simili alle forme d'onda audio reali corrispondenti a un determinato input, come spettri di potenza o coefficienti di predizione lineare (LPC). Questo viene fatto attraverso la competizione tra un generatore e un discriminatore all'interno della GAN. Il generatore cerca di

generare forme d'onda più realistiche possibili ispirandosi a quelle in input, il discriminatore cerca di distinguere gli audio reali da quelli sintetici. In questo modo il generatore cercherà di migliorare la qualità di generazione mentre il discriminatore imparerà ancor di più a smascherare gli originali da quelli realistici, tutto ciò a vantaggio della qualità di questa famiglia di modelli. Tra i vocoder di questo tipo possiamo citare MelGAN, GAN-TTS, WaveGAN, HiFi-GAN, Parallel WaveGAN or Multi-Band MelGAN.

3.2 Dataset utilizzato

Il dataset utilizzato nello studio è molto simile a quello descritto nell'articolo di SpecRNet e consiste in due classi di registrazioni audio: una classe sintetica (spoof) e una classe originale (bonafide). La parte del dataset degli audio sintetici include le registrazioni presenti nel dataset WaveFake [2]. Questa raccolta di dati è composta da 117.985 audio in formato 16-bit PCM wav.
I dati sintetici sono stati generati utilizzando alcuni vocoder menzionati in precedenza, che sono stati alimentati da due dataset di registrazioni originali: LJSPEECH e JSUT. Dal dataset LJSPEECH sono state prese 13.000 clip audio di una voce femminile che legge estratti di 7 libri diversi. Queste registrazioni sono state effettuate utilizzando il microfono di un MacBook e hanno una durata complessiva di 24 ore.
Il dataset JSUT include 37.177 clip audio in lingua giapponese, che comprendono frasi, onomatopee, parafrasi, suffissi e varie raccolte di parole. Le registrazioni sono state effettuate da una speaker donna in una stanza anecoica e hanno una durata totale di 10 ore.
Entrambi questi dataset di registrazioni originali sono stati utilizzati per generare gli audio sintetici presenti nella parte del dataset dedicata alle registrazioni spoof.
Ogni set di cui si compone la raccolta dati di WaveFake fa riferimento ad un vocoder e ad un input differente: ad esempio la raccolta "jsut_multi_band_melgan" include registrazioni prodotte con la rete Multi-band MelGAN partendo dal subset di JSUT, la raccolta "ljspeech_full_band_melgan" si compone di audio sintetici prodotti dalla rete Full-band MelGAN usando il dataset LJSPEECH, e così via.

4 Data preparation e Data augmentation

Nel contesto specifico di questo progetto, che si concentra sull'analisi audio e sul riconoscimento tra audio sintetico e originale, la fase di Data Preparation riveste un'importanza cruciale, poiché il modo in cui viene gestita può avere un impatto significativo sulle prestazioni e sull'accuratezza del modello.
Nella fase iniziale di Data Preparation, sono state effettuate due attività principali: la rimozione del silenzio e lo splitting dei file audio per ottenere segmenti di lunghezza fissa pari a 4 secondi.
Per rimuovere le parti di silenzio da un file audio, è stata utilizzata la funzione ***trim*** della libreria Librosa. Questa funzione serve per individuare le regioni non silenziose di un segnale audio e restituire una versione troncata del segnale originale contenente solo queste regioni.
Dopo la fase di Data Preparation, è stata eseguita la fase di Data Augmentation, durante la quale sono state applicate diverse tecniche di trasformazione degli audio. Queste trasformazioni sintetiche hanno consentito di generare nuovi sample di dati con l'obiettivo di aumentare la diversità e la quantità dei dati di addestramento.
La fase di Data Augmentation ha coinvolto esclusivamente gli audio bonafide, poiché questa classe di audio era quella meno rappresentata in tutti i dataset utilizzati. Questo approccio ha consentito di aumentare il numero di audio bonafide e di bilanciare il dataset in modo più equilibrato.
Le attività di Data Augmentation svolte sono le seguenti:

1. Aggiunta di rumore al file audio;

2. Stretch del file audio,ovvero la sua durata temporale è stata ridotta del 20%. Di conseguenza, l'audio verrà riprodotto a una velocità più elevata, ma la sua tonalità o frequenza rimarranno invariate. Per fare ciò è stata utilizzata la funzione ***time_stretch*** della libreria Librosa;

3. Shift del file audio, ovvero è stato spostato l'intero segnale audio lungo l'asse temporale in avanti o indietro in modo casuale;

4. Pitch del file audio, ovvero è stata modificata l'altezza dei semitoni del contenuto audio, senza però influire la durata temporale.

Successivamente, dopo aver normalizzato ed aumentato il dataset iniziale, sono stati estratti gli MFCC e gli LFCC corrispondenti da ciascun file audio.
Per estrarre gli MFCC è stata utilizzata la funzione ***mfcc*** della libreria Librosa mentre per estrarre gli LFCC è stata utilizzata la tecnica classica di ***Short-time Fourier transform***.

5 Architettura del Modello

Il modello utilizzato è una variante implementata in Keras di un modello già chiamato SpecRnet, il quale è stato originariamente sviluppato in PyTorch.
Pertanto, in questa sezione verrà descritto il modello implementato in Keras, che ha una struttura pressoché identica al modello preesistente SpecRnet sviluppato in PyTorch.
Di seguito viene riportato lo schema che rappresenta la struttura generale del modello di rete neurale.

Il modello prende in input la rappresentazione MFCC o LFCC di un segnale audio. L'input della rete viene prima normalizzato utilizzando la batch normalization, seguita dalla funzione di attivazione SeLU. Successivamente, il modello è composto da 3 Residual Blocks contenenti due layer convoluzionali preceduti da un layer di batch normalization e un layer con funzione di attivazione LeakyReLU. I layer di convoluzione sono definiti secondo la conven-

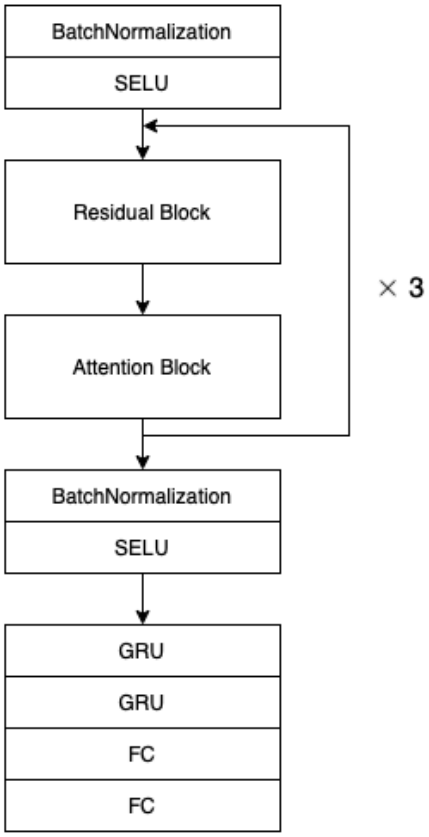

Figure 1: Struttura del modello Keras

zione: Conv2D(output channels, kernel size, input channels).
La normalizzazione del batch e la funzione di attivazione LeakyReLU vengono applicate all'input corrispondente solamente dal secondo Residual Block in poi, poiché l'input iniziale viene già normalizzato e sottoposto alla LeakyReLU all'inizio, prima del primo Residual Block.

5.1 Residual Block

Come si può vedere nella Figura 2, all'interno del Residual Block viene utilizzato un ulteriore layer convoluzionale con una dimensione di kernel pari a 1 nel percorso di identità residua, che viene poi sommato all'output dell'ultimo layer convoluzionale. La somma dei due layer convoluzionali è una tecnica nota come "skip connection" o "residual connection". Quando si sommano due layer convoluzionali, l'obiettivo è creare un collegamento diretto tra l'input e l'output del Residual Block. Ciò consente al gradiente di propagarsi più facilmente attraverso il Residual Block senza subire una forte attenuazione, risolvendo così il problema della scomparsa del gradiente durante l'addestramento della rete. Inoltre, la somma dei due layer può aiutare a preservare le informazioni dell'input originale, consentendo al modello di apprendere le differenze o i "residui" tra l'input e l'output.
È importante notare che la somma dei due layer convoluzionali può essere eseguita solo se le dimensioni dell'input e dell'output sono compatibili. Pertanto, è per questo motivo che viene utilizzato un layer convoluzionale con un kernel size pari a 1 per sincronizzare il numero di canali tra il percorso di identità e il percorso principale prima di effettuare la somma.
Nell'ultimo Residual Block, a causa del numero costante di canali (64), non è necessaria alcuna sincronizzazione tra il percorso di identità e il percorso principale, quindi in questo caso non viene applicato il layer convoluzionale aggiuntivo.
Successivamente, ogni blocco è seguito da un layer di max pooling bidimensionale, un Attention Block e un altro layer di max pooling bidimensionale. Il passaggio in avanti attraverso il Residual Block e l'Attention Block è illustrato nella Figura 2.

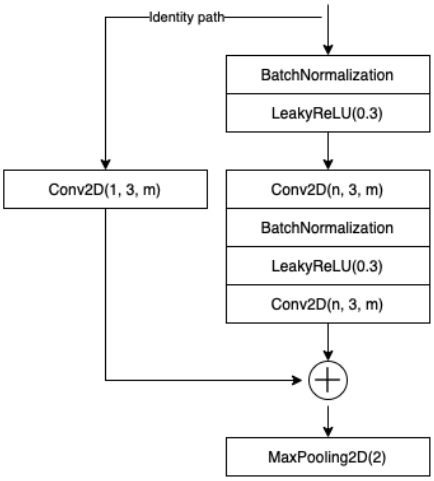

Figure 2: Struttura del Residual Block

5.2 Attention Block

L'Attention Block è una componente che è stata utilizzata per focalizzare l'attenzione del modello su determinate caratteristiche o regioni rilevanti dei dati in ingresso. L'Attention Block è composto da tre parti principali:

1. Global Avarage Pooling Layer: Questo strato esegue un'operazione di pooling globale sull'output dell'ultima convoluzione della rete. L'obiettivo del pooling globale è quello di ridurre la dimensione spaziale delle features map a una singola dimensione. e catturare le informazioni globali dell'immagine;

2. Dense layer: Il layer Dense con una funzione di attivazione ReLU viene utilizzato per proiettare il vettore di feature map appiattito in uno spazio di dimensione n in output. Questo layer svolge il compito di "attenzione" sulla rappresentazione delle features, selezionando le informazioni più rilevanti;

3. Function Sigmoid: La funzione di attivazione Sigmoid viene applicata al vettore di output del layer Dense per ottenere un valore compreso tra 0 e 1 per ogni feature map di output. Utilizzando questa funzione, viene calcolata l'importanza relativa di ciascun valore presente nella feature map.

Successivamente, viene eseguita una moltiplicazione elemento per elemento tra l'input originale in ingresso e il tensore restituito in output dal meccanismo di attenzione. Questa moltiplicazione viene eseguita per applicare l'attenzione alle feature map originali, dando maggiore importanza alle regioni selezionate dall'attenzione.

Infine, viene applicato un layer di MaxPooling2D per ridurre la dimensione spaziale delle feature map risultanti, consentendo di conservare le caratteristiche più rilevanti riducendo al contempo la dimensionalità.
L'architettura termina con due layer GRU bidirezionali, seguiti da due layer completamente connessi (due Dense Layers rispettivamente da 128 e 1 nuerone). Il modello restituisce un singolo valore in un intervallo compreso tra 0 e 1 che, dopo l'applicazione di una soglia, indica se l'input è spoof o bonafide.

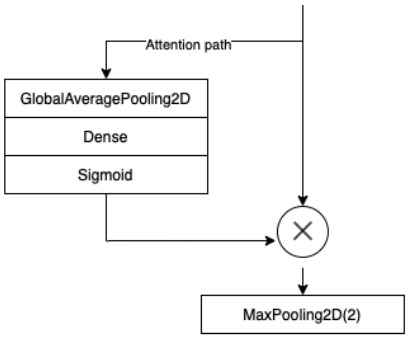

Figure 3: Struttura dell'Attention Block

6 Training del modello

Sono state addestrate due istanze del modello per confrontare le performance di addestramento utilizzando due diverse tecniche di estrazione delle caratteristiche audio, ovvero MFCC e LFCC.
Il dataset utilizzato nei benchmarks è composto da 164021 campioni spoof e 171124 campioni bonafide.
L'obiettivo era quello di addestrare i modelli per un periodo di 10 epoche, ciascuna conclusa con una validazione su un set di test. Tuttavia, analizzando la val_loss dei rispettivi modelli, è stato osservato che dopo un certo numero di epoche (es. 3) la val_loss smetteva di diminuire e iniziava ad aumentare. Pertanto durante la fase di validazione è stata scelta la versione del modello che presentava il valore di val_loss più basso raggiunto durante le epoche di addestramento.
Per i set di addestramento, validazione e test, è stata seguita la convenzione standard di denominazione "train", "validation" e "test" (o "eval").
Ognuno di questi tre sottoinsiemi conteneva campioni di ogni tipo di attacco ed è stato suddiviso rispettivamente con un rapporto del 70:15:15 per garantire che ogni set fosse rappresentativo di tutte le tipologie di attacco. Per garantire che i risultati siano rappresentativi delle prestazioni del modello sul dataset, entrambi i modelli sono stati allenati, validati e testati utilizzando tre differenti

seeds (1, 42, 77).
L'addestramento è stato basato sull'ottimizzatore Adam con batch_size pari a 16, utilizzando la funzione di perdita binary-cross-entropy. Per evitare errori di memoria durante l'addestramento del modello dovuti alla grande quantità di dati in input, è stato implementato un sistema di generazione di batch in tempo reale.
Il numero di parametri allenanti per il modello che utilizza gli MFCC è 303,769 mentre quello che utilizza gli LFCC è 352,921.

6.1 Tempistiche della fase di training

Al termine della fase di addestramento sono state calcolate le tempistiche di entrambi i modelli. È emerso che il tempo impiegato per eseguire lo stesso numero di epoche è pressoché lo stesso (es 12 ore = 3 epoche).

7 Risultati e considerazioni finali

7.1 Metriche utilizzate

Al fine di ottenere un confronto oggettivo con il modello descritto dagli autori dell'articolo, sono state misurate le stesse metriche utilizzate dagli autori stessi per valutare i modelli, ovvero:

1. **Mean EER (%)** : L'Equal Error Rate (EER) viene comunemente utilizzato per la rilevazione di DeepFake e tasks anti-spoofing. In particolare, mostriamo l'EER come percentuale;

2. **Mean AUC**: L'Area Under Curve (AUC) della curva Receiver Operating Characteristic (ROC) è una metrica che riflette una corrispondenza statistica ed è comunemente utilizzata in task di classificazione binaria;

7.2 Risultati

Per misurare la Precision, la Recall e F1-score è stata utilizzata la funzione ***classification_report*** della libreria Scikit-learn.
Le metriche di classificazione di Scikit-learn dei modelli basati sulle due tecniche di estrazione delle feature allenati, validati e valutati sui rispettivi set generati con seed 77, hanno ottenuto i seguenti risultati:

Classes	Precision	Recall	F1-score	Support
Spoof	1.00	0.99	1.00	20133
Bonafide	0.98	0.99	0.99	5083

Table 1: Risultati delle predizioni del modello migliore basato su MFCC

Classes	Precision	Recall	F1-score	Support
Spoof	1.00	1.00	1.00	20133
Bonafide	1.00	0.99	0.99	5083

Table 2: Risultati delle predizioni del modello migliore basato su LFCC

I valori delle metriche EER (in percentuale) e ROC medie dei rispettivi modelli sono riportate di seguito:

EER (std)	AUC (std)
0.85 (0.00209)	99.94 (0.00034)

Table 3: EER medio (%) e AUC delle predizioni del modello allenato con gli MFCC.

EER (std)	AUC (std)
0.46 (0.1593)	99.992 (0.0100)

Table 4: EER medio (%) e AUC delle predizioni del modello allenato con gli LFCC.

PWG	JSUT-MB-MG	JSUT-PWG	LJ-FB-MG	LJ-HG	LJ-MG	LJ-MGL	LJ-MB-MG	LJ-PWG	LJ-WG
1.36	0.87	2.11	0.94	1.81	0.81	0.83	0.67	1.12	0.83

Table 5: EER(%) medio delle predizioni del modello allenato e validato senza la rispettiva tecnica di spoofing con gli MFCC

PWG	JSUT-MB-MG	JSUT-PWG	LJ-FB-MG	LJ-HG	LJ-MG	LJ-MGL	LJ-MB-MG	LJ-PWG	LJ-WG
1.26	0.96	3.01	0.41	0.41	0.89	0.39	0.67	0.28	0.39

Table 6: EER(%) medio delle predizioni del modello allenato e validato senza la rispettiva tecnica di spoofing con gli LFCC

7.2.1 Considerazioni

Nell'articolo da cui è stato tratto ispirazione per il progetto, si evidenzia l'interesse di valutare le prestazioni del modello utilizzando un'altra rappresentazione dei campioni audio. Di conseguenza, l'obiettivo del progetto è stato confrontare i risultati ottenuti da un singolo modello allenato utilizzando due diverse tecniche di estrazione audio: MFCC e LFCC. Questo confronto permette di valutare l'impatto delle diverse rappresentazioni audio sulle prestazioni del modello e di determinare quale tecnica sia più efficace per il riconoscimento degli audio sintetici nel contesto degli attacchi Text-To-Speech.

Nelle tabelle 3 e 4 vengono riportati i risultati ottenuti durante la fase di testing. Da questi risultati emerge che il modello addestrato con gli LFCC sembra aver ottenuto prestazioni migliori rispetto a quello addestrato con gli MFCC.
Un altro studio condotto ha coinvolto l'analisi di come varie tecniche di DeepFake (attacchi) influenzino le prestazioni dei modelli finali. Sono stati condotti 10 train differenti escludendo uno tipo di attacco alla volta, e successivamente è stata condotta la procedura di test considerando l'intero dataset. Le tabelle 5 e 6 riassumono i risultati ottenuti. In effetti, anche in questo caso il modello allenato con gli LFCC sembra offrire risultati migliori nella maggior parte dei test svolti.
Come specificato anche dagli autori, questa anlisi degli attacchi circoscritti ha permesso di ottenere informazioni sulle relazioni tra gli attacchi specifici e l'efficacia dei metodi investigati.
I risultati ottenuti possono essere interpretati nel seguente modo:

1. **Valore alto di EER**: il determinato attacco è stato rilevato correttamente nello scenario di base, ovvero si può considerare come un "attacco facilmente rilevabile" in quanto la sua assenza nello scenario di attacchi limitati peggiora le prestazioni;

2. **Valore basso di EER**: un valore basso di EER suggerisce che un dato attacco è difficile da rilevare, in quanto la sua rimozione nello scenario di attacchi limitati non influenza in modo significativo le prestazioni del modello.

Nel caso di un EER gli autori suggeriscono che potrebbe essere opportuno addestrare metodi specializzati che identifichino correttamente (e con alta efficacia) questo attacco specifico e, quando viene rilevato, il risultato dei metodi specializzati sostituisce il modello generale di rilevamento di DeepFake.
In conclusione, si può dire che l'utilizzo degli LFCC come tecnica di estrazione audio potrebbe essere vantaggioso per il compito considerato, poiché il modello addestrato con questa rappresentazione degli auddio sembra avere una migliore capacità di discriminazione tra le tecniche di deepfake.

References

[1] Ketan Doshi. Why mel spectrograms perform better. https://towardsdatascience.com/audio-deep-learning-made-simple-part-2-why-mel-spectrograms-perform-better-aad889a93505, 2021. [Online; accessed 19-February-2021].

[2] Joel Frank and Lea Schönherr. Wavefake: A data set to facilitate audio deepfake detection, 2021.

[3] Piotr Kawa, Marcin Plata, and Piotr Syga. Specrnet: Towards faster and more accessible audio deepfake detection, 2022.