



UNIVERSITÀ
degli STUDI
di CATANIA

Steganografia applicata a segnali audio

Fabio Merola W82000188.

Professori: Filippo Stanco, Dario Allegra.

Dipartimento: D.M.I. Informatica LM-18 A.A. 2019/2020

Data documento: 3 Maggio 2020.

Indice

1	Introduzione	3
2	Sviluppo script	3
2.1	Da testo ad audio	3
2.2	Da audio a testo	5
2.3	Attacchi al supporto audio	7
2.3.1	Filtro di Butterworth	7
2.3.2	Conversione in diversi formati audio	8
2.3.3	Rumore addizionale	8
2.4	Algoritmo di valutazione	9
3	Fase di test	10
4	Interpretazione dei risultati	12
4.1	Sviluppi futuri	15

1 Introduzione

Lo scopo di questo elaborato è quello di riuscire ad iniettare del testo all'interno dello spettro di un segnale audio, allo scopo ultimo di utilizzarlo come supporto steganografico. Inoltre, allo scopo di sviluppare un ambiente completo, si è optato anche per la creazione di un metodo di estrazione del testo. Questi, sono poi stati testati a scopo di analisi in funzione di più variabili e sottomessi a diversi attacchi nell'intento di disturbare la fase di decodifica. Il progetto è quindi diviso in tre fasi principali:

- Sviluppo script;
- Fase di test;
- Interpretazione dei risultati.

2 Sviluppo script

Tutti gli script sono sviluppati in Python e sono disponibili, insieme alle analisi principali su GitHub^[1]. Le librerie utilizzate sono alle seguenti versioni:

Libreria	Versione
Scipy	1.1.0
Matplotlib	2.2.5
Numpy	1.15.2
PIL	5.3.0
PyTesseract	0.3.4
PyDub	0.23.1
Levenshtein	0.12.0

2.1 Da testo ad audio

Tutto inizia da una stringa di testo in input che inizialmente viene stampata su un'immagine a sfondo nero dalla funzione:

```
def text_on_img(
    filename ,
    text ,
    size ,
    extension=".pbm" ,
    freq="HF" ,
    font_size=25,
    font='FreeSansBold.ttf'
)
```

che come intuibile permette anche di settare, per l'immagine risultante, vari parametri quali il nome, la dimensione, il font e la sua dimensione, l'estensione ma soprattutto la fascia di frequenze in cui questo testo deve essere iniettato. Questi saranno fondamentali nella fase di test successiva.

Esempi di immagini risultanti da questa funzione sono:



Fig. 1: Testo iniettato nelle basse frequenze(`freq="LF"`).

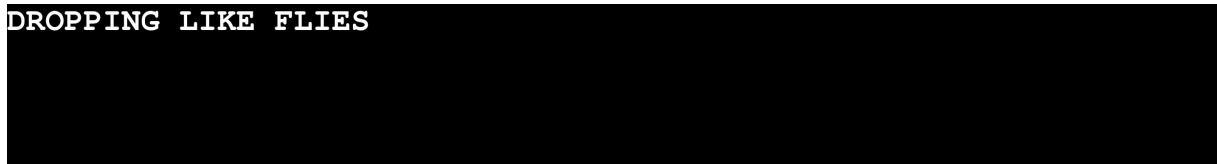


Fig. 2: Testo iniettato nelle alte frequenze(`freq="HF"`).

Il procedimento appena descritto è risultato necessario per ottenere il segnale audio in quanto è possibile, tramite l'inversa della trasformata discreta di Fourier, utilizzare l'immagine come spettro di un segnale per trasformarla in suono. Questo avviene nella funzione:

```
def make_wav_from_img(  
    image_filename,  
    sampling_freq=44100  
)
```

Infine si attua, nella funzione illustrata di seguito, una semplice sovrapposizione del nuovo segnale audio ottenuto con i passaggi precedenti, al segnale audio originale che useremo come portante del messaggio.

```
def wav_overlay(  
    wav_msg_path,  
    original_audio=ORIGINAL_AUDIO_PATH  
)
```

2.2 Da audio a testo

Questa fase inizia calcolando lo spettro del segnale audio in input tramite la funzione:

```
def calc_specto(  
    wav_overlay_paths,  
    img_wav_paths=[]  
)
```

Ogni spettro viene quindi plottato allo scopo di acquisirne l'immagine, ottenendo qualcosa del tipo:

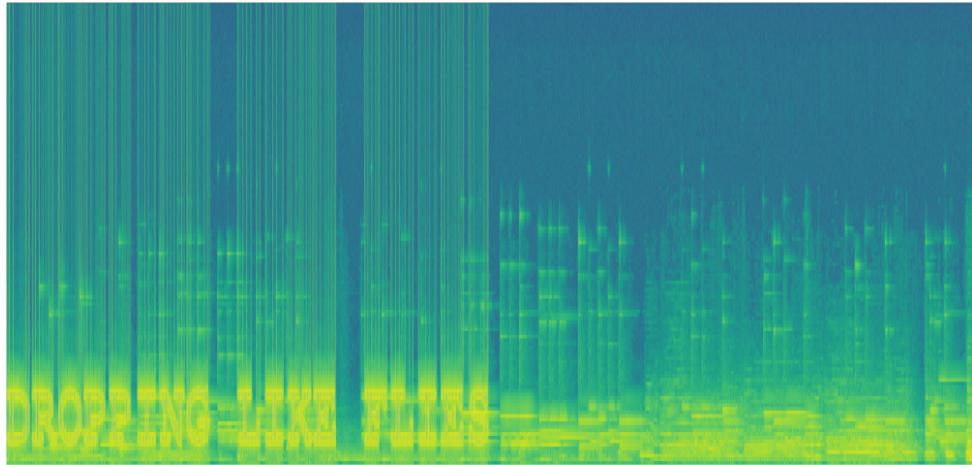


Fig. 3: Spettro di un audio risultante dalla fase precedente con testo iniettato nelle basse frequenze.

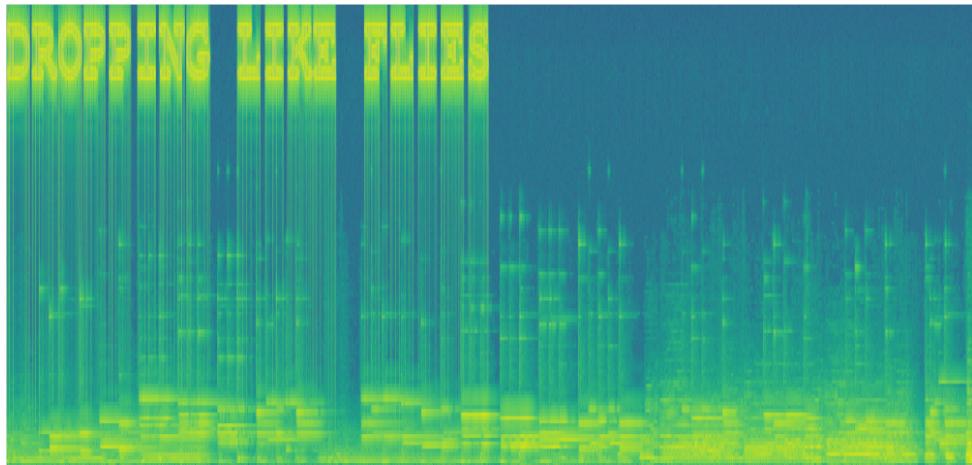


Fig. 4: Spettro di un audio risultante dalla fase precedente con testo iniettato nelle alte frequenze.

Immagini del genere, però, difficilmente vengono interpretate correttamente dall'OCR, motivo per il quale si è optato per un elaborazione post-processing nella funzione:

```
def images_post_processing()
```

Che eseguisse, nell'ordine, le seguenti operazioni:

1. Conversione in scala di grigi;
2. Prima sogliatura per scremare i dettagli inutili;
3. Dilatazione;
4. Erosione;
5. Seconda sogliatura per rifinire il risultato ottenuto dalle operazioni precedenti;
6. Ridimensionamento(empiricamente si è verificato che l'OCR si comporta meglio con immagini 2000[px]x1000[px]);
7. Terza sogliatura per eliminare gli artefatti uscenti dall'interpolazione del ridimensionamento;
8. Piccolo ritaglio per eliminare bordi provenienti dalla fase di plotting.

Questo ci permette di ottenere immagini simili alle seguenti:



Fig. 5: Immagine risultante dalla fase di post-processing con testo iniettato nelle basse frequenze.



Fig. 6: Immagine risultante dalla fase di post-processing con testo iniettato nelle alte frequenze.

Infine si mandano le immagini all'OCR che darà in output una stringa, con la seguente funzione che sfrutta la libreria `pytesseract` che funge da wrapper per l'OCR Tesseract^[2].

```
def images_in_ocr()
```

2.3 Attacchi al supporto audio

2.3.1 Filtro di Butterworth

Il primo dei tre attacchi al supporto audio, attuati per verificare la robustezza, è l'applicazione di un filtro di Butterworth in tre varianti:

- Passa-basso : rimuove tutte le frequenze superiori ai 7350[Hz];
- Passa-banda : rimuove tutte le frequenze inferiori ai 7350[Hz] e superiori ai 14700[Hz];
- Passa-alto : rimuove tutte le frequenze inferiori ai 14700[Hz].

Il polinomio di Butterworth usato per tutte le varianti è di ordine pari a 5 e viene implementato nella funzione:

```
def apply_butter_bandpass_filter(
    file_path,
    butter_filter
)
```

Questo porta ad ottenere spettri di questo genere:

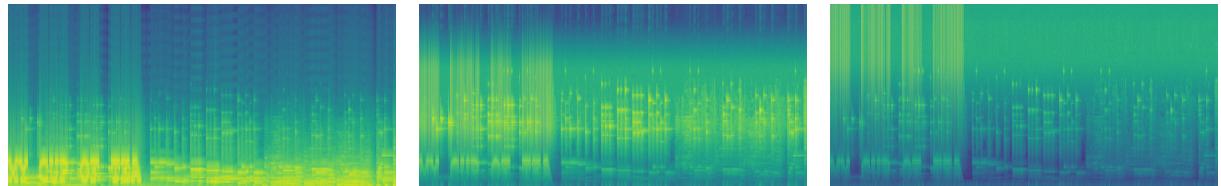


Fig. 7: Segnale a cui è stato applicato un filtro di Butterworth(passa-basso), con testo iniettato nelle basse frequenze.

Fig. 8: Segnale a cui è stato applicato un filtro di Butterworth(passa-banda), con testo iniettato nelle basse frequenze.

Fig. 9: Segnale a cui è stato applicato un filtro di Butterworth(passa-alto), con testo iniettato nelle basse frequenze.

2.3.2 Conversione in diversi formati audio

Il secondo metodo testato riguarda la conversione dal formato file audio WAV(WAVEform audio file format), quindi non compresso, ad altri che notoriamente applicano algoritmi di compressione "lossy"(con perdita di dati), quali:

- MP3(Moving Picture Expert Group-1/2 Audio Layer 3)[.mp3];
- Ogg Vorbis(Ogg Media compressed by Vorbis algorithm)[.ogg];
- AAC(Advanced Audio Coding)[.aac].

Questo avviene nella funzione:

```
def change_ext(file_path, ext)
```

Di seguito un esempio di paragone tra gli spettri ottenuti con i differenti formati:

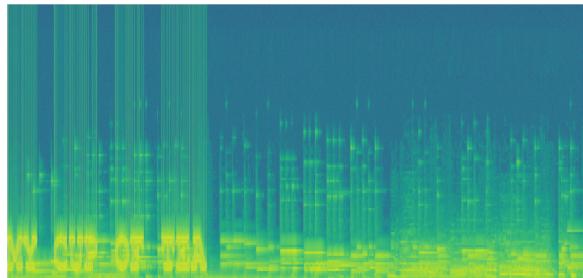


Fig. 10: Spettro audio originale WAV(Dimensione: 2,6 MB)

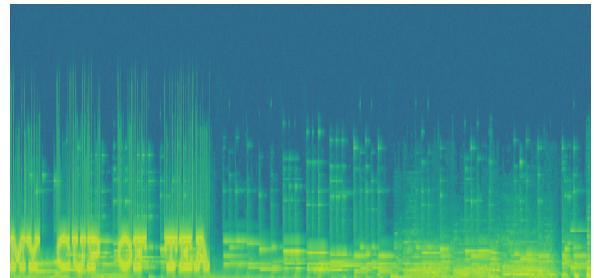


Fig. 11: Spettro audio convertito in MP3(Dimensione: 232,6 kB)

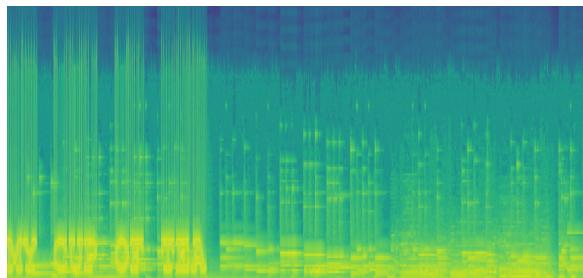


Fig. 12: Spettro audio convertito in Ogg Vorbis(Dimensione: 259,9 kB)

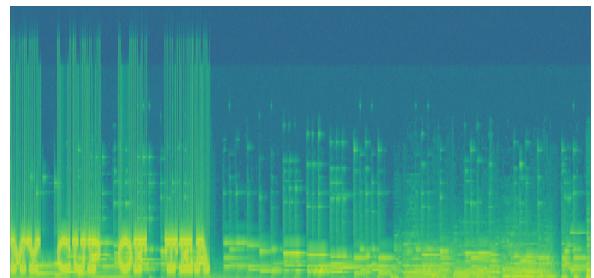


Fig. 13: Spettro audio convertito in AAC(Dimensione: 262,2 kB)

2.3.3 Rumore addizionale

Il terzo ed ultimo attacco implementato, prevede l'aggiunta controllata di rumore bianco Gaussiano. Controllata perché viene aggiunto sulla base di un SNR(Signal to Noise Ratio) dato in input alla funzione:

```
def add_white_noise(file_path, SNR)
```

Esempi di questo attacco sono resi disponibili di seguito:

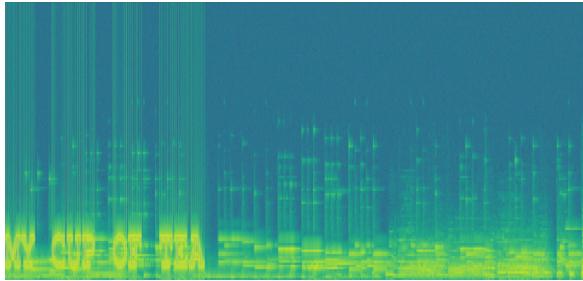


Fig. 14: Spettro audio con rumore bianco Gaussiano, SNR: 50

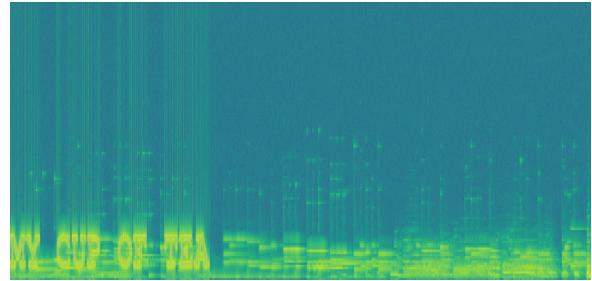


Fig. 15: Spettro audio con rumore bianco Gaussiano, SNR: 20

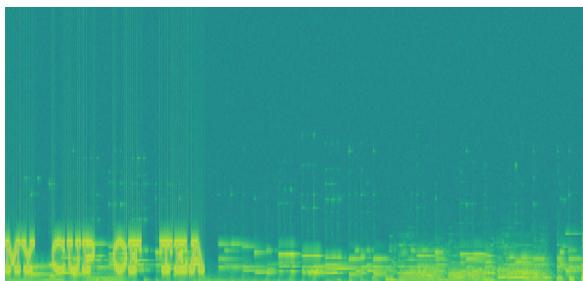


Fig. 16: Spettro audio con rumore bianco Gaussiano, SNR: 10

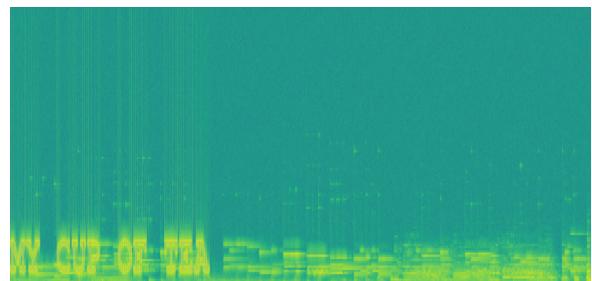


Fig. 17: Spettro audio con rumore bianco Gaussiano, SNR: 5

2.4 Algoritmo di valutazione

Allo scopo di valutare la similarità tra due stringhe, in particolare tra quella ottenuta dall'OCR e quella iniettata inizialmente nello spettro, si è optato per l'utilizzo dell'algoritmo di Levenshtein tramite apposita libreria Python, disponibile nello script:

```
levenshtein_similarity_ratio.py
```

Questo misura la distanza, in termini di modifiche, che si interpone tra due stringhe per poi calcolarne un indice di similarità.

3 Fase di test

Questa fase è caratterizzata dalla scelta degli input e dalla computazione degli output, nel dettaglio, per rendere meno dipendenti possibile i risultati dalla tipologia di testo inserito si è optato per usare dieci testi diversi:

- "*My Cup of Tea*";
- "*Right Out of the Gate*";
- "*Shot In the Dark*";
- "*Rain on Your Parade*";
- "*Dropping Like Flies*";
- "*No Ifs, Ands, or Buts*";
- "*Everything But The Kitchen Sink*";
- "*Quality Time*";
- "*Tug of War*";
- "*Jig Is Up*".

Ogni testo è stato trattato come segue:

- Iniettato nelle alte e nelle basse frequenze con tutti i caratteri in maiuscolo;
- Trascritto con 3 font diversi:
 - Mono;
 - Sans;
 - Serif.

Ognuno dei quali utilizzato in versione normale ed in grassetto e con dimensioni pari a:

- 25 pt;
- 50 pt;
- 75 pt;
- 100 pt.

Per ogni immagine, derivante dalla combinazione delle precedenti variabili, si è optato per l'archiviazione in tre differenti formati:

- JPEG(Joint Photographic Experts Group)[.jpeg];
- PBM(Portable Bitmap)[.pbm];

- PNG(Portable Network Graphics)[.png];

Questa combinazione di variabili ha portato alla creazione ed all'analisi di un dataset di 1440 campioni, di cui i migliori otto sono stati testati contro gli attacchi proposti, con i seguenti input:

- Filtri di Butterworth:
 - Low-pass (fino a 7350 Hz);
 - Band-pass (da 7350 Hz fino a 14700 Hz);
 - High-pass (da 14700 Hz).
- Conversione in formato "lossy":
 - AAC;
 - MP3;
 - Ogg Vorbis.
- Aggiunta di rumore bianco Gaussiano con SNR pari a:
 - 5;
 - 10;
 - 20;
 - 50.

Il che ha portato all'aggiunta di altri 80 campioni al dataset iniziale.
Il dataset è reso disponibile su Google Drive^[3].

4 Interpretazione dei risultati

Partendo dall'analisi del font migliore utilizzato, è facile intuire dal grafico successivo che il FreeMono in grassetto è quello che ha dato risultati migliori:

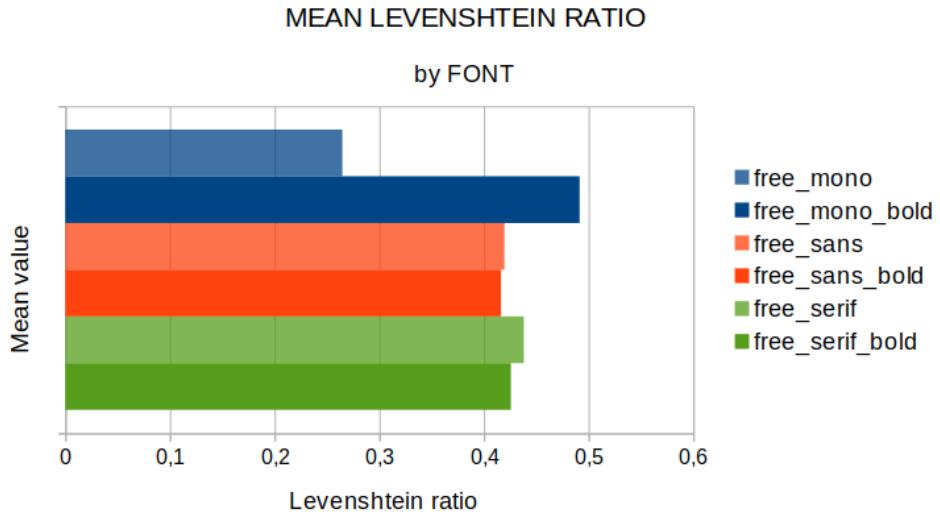


Fig. 18: Grafico comparativo dei font dipendentemente dall'indice di Levenshtein medio.

Nel dettaglio, questi i risultati dipendenti anche dalla dimensione:

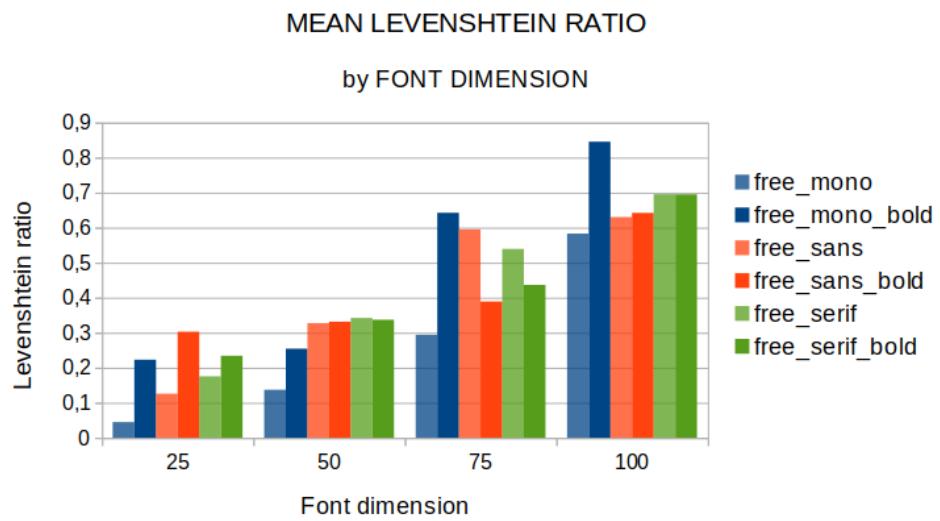


Fig. 19: Grafico comparativo dei font dipendentemente dall'indice di Levenshtein medio e dalla dimensione del font.

Dove si evince che il font citato poc'anzi non è sempre ottimale ma solo per le dimensioni più grandi, dimensioni per le quali tutti i font risultano essere più comprensibili per l'OCR.

Per quanto invece riguarda le altre variabili in gioco, abbiamo che non esiste differenza tra i risultati ottenuti da un’immagine con estensione PNG ed una PBM, inoltre la differenza tra queste e quelle in JPEG è davvero minima, anche la differenza tra testi iniettati nelle alte e nelle basse frequenze è minima, a favore delle ultime.

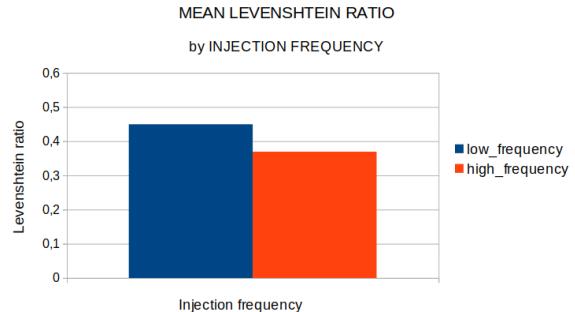
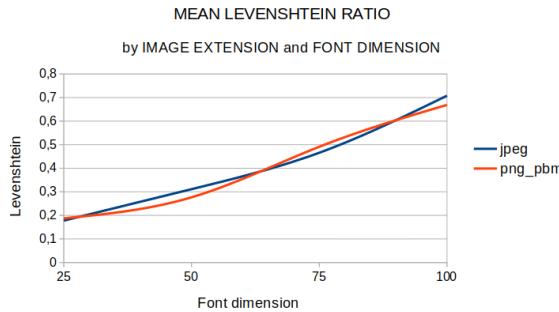


Fig. 20: Grafico comparativo delle estensioni utilizzate dipendentemente dall’indice di Levenshtein medio e dalla dimensione del font.

Fig. 21: Grafico comparativo dei risultati dipendentemente dall’indice di Levenshtein medio e dalla frequenza in cui è stato iniettato il testo.

Per la fase degli attacchi, di seguito, sono riportati dei grafici riassuntivi che espletano la risposta, in termini di percentuale di informazione persa, dei campioni sottoposti ai test dipendentemente dalla dimensione del font(Fig. 22) e dalla frequenza in cui il testo è stato iniettato(Fig. 23).

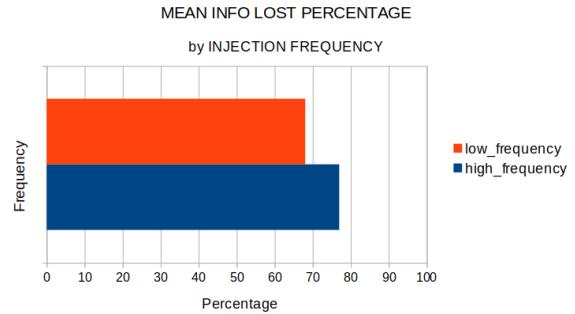
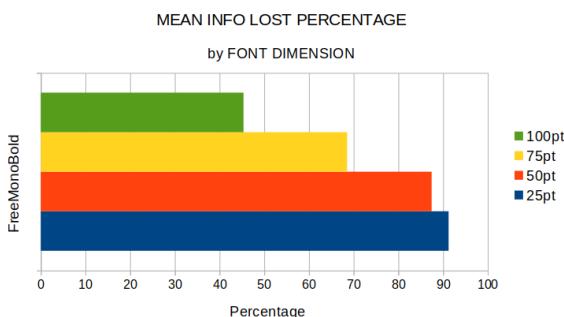


Fig. 22: Grafico comparativo delle percentuali di informazioni mediamente perse dipendentemente dalla dimensione del font.

Fig. 23: Grafico comparativo delle percentuali di informazioni mediamente perse dipendentemente dalla frequenza in cui il testo è stato iniettato.

Come ci si poteva aspettare, si nota subito che ad una dimensione dei caratteri inferiore corrisponda una maggiore perdita di informazioni. Allo stesso modo, come visto nei grafici precedenti, iniettare il testo nelle basse frequenze risulta essere la scelta migliore nonostante l’aggiunta di un contributo nelle stesse, finisce per essere uditivamente disturbante.

Di seguito una panoramica generale degli effetti dei singoli attacchi:

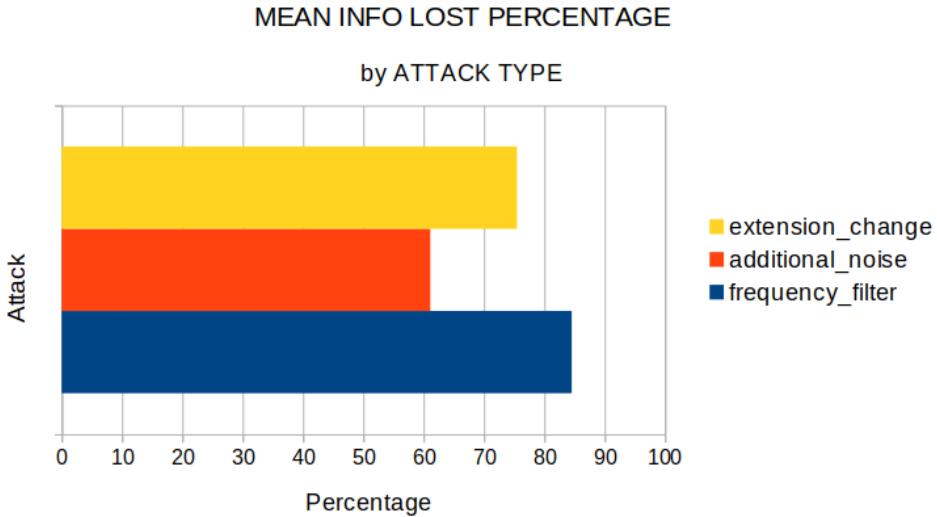


Fig. 24: Grafico comparativo degli attacchi dipendentemente dalla percentuale di informazioni perse.

L'attacco che risulta essere più ostacolante è chiaramente quello che riguarda la cancellazione di parte del segnale, non solo perché potrebbe eliminare la parte dello spettro contenente il testo, ma anche perché non esistendo un filtro passa-banda ideale, qualora il messaggio fosse all'interno della banda concessa, non è detto che risulti pienamente comprensibile ad un occhio non umano.

Nel dettaglio, questi i risultati per ogni tipologia di attacco implementata:

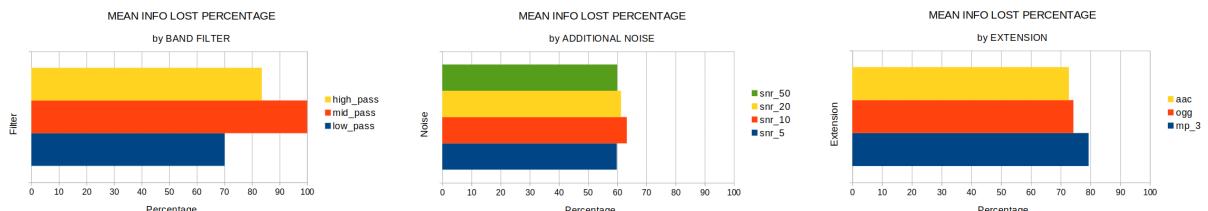


Fig. 25: Grafico comparativo degli attacchi basati su filtro di Butterworth dipendente- mente dalla percentuale di informazioni perse.

Fig. 26: Grafico compar- ativo degli attacchi basati sull'aggiunta di rumore di- pendente dalla percen- tuale di informazioni perse.

Fig. 27: Grafico compar- ativo degli attacchi basato sul cambio di estensione di- pendente dalla percen- tuale di informazioni perse.

Nel grafico in Fig. 25 si verifica esattamente quanto detto prima, in particolare per le alte frequenze.

Per il grafico in Fig. 26 si nota come non ci siano molte differenze tra un rumore aggiunto con valore di SNR pari a 5 ed uno pari a 50 probabilmente perché, considerata la tipologia di rumore utilizzata, il processo si traduce in un aumento di contributo assolutamente omogeneo nello spettro.

Come da immagini di test, il grafico in Fig. 27 mostra come le tre estensioni utilizzino algoritmi di compressione diversi, che vanno spesso ad intaccare le frequenze più alte essendo raramente udibili. Questo porta ad avere risultati diversi dovuti esclusivamente ai campioni utilizzati con testo iniettato nelle alte frequenze.

4.1 Sviluppi futuri

Tra i tanti eventuali sviluppi futuri i più interessanti potrebbero comprendere:

- L'utilizzo di algoritmi crittografici al fine di rendere il messaggio indecifrabile per uno sconosciuto;
- L'utilizzo, dopo attente analisi, di uno pseudo-alfabeto(ad esempio per i test fatti si è optato per l'utilizzo di lettere esclusivamente maiuscole) che risulti più robusto ai problemi evidenziati in questo elaborato;
- L'implementazione di questo elaborato in versione "real-time" da poter utilizzare, ad esempio, all'interno di una classica telefonata o di una VoIP.

Riferimenti

[1] Repository GitHub del progetto: Multimedia_Project.
[https://github.com/Dophy6/Multimedia_Project].
Ultimo accesso: 8 Giugno 2020.

[2] Repository GitHub dell'OCR: Tesseract.
[<https://github.com/tesseract-ocr/tesseract>].
Ultimo accesso: 8 Giugno 2020.

[3] Dataset del progetto: MM_Project_DATASET.
[https://drive.google.com/drive/folders/MM_Project_DATASET].
Ultimo accesso: 8 Giugno 2020.