בס"ד

הפקולטה להנדסה
ע"ש אלכסנדר קופקין
אוניברסיטת בר־אילן

Reverberation time estimation by deep learning for echo cancellation usage

שערוך זמן הדהוד על ידי למידה עמוקה לצורך ביטול מיטבי של הד

מגישות:

עדי כהן

דניאל לוי

מנחים:

דוקטור עופר שוורץ

מר אייל שוורץ

אחראי אקדמי:

פרופסור שרון גנות

תוכן עניינים

רקע לפרויקט 3
מטרת הפרויקט 3
חלק א: מימוש המסננים לביטול ההד כאשר זמן ההדהוד ידוע 4
שלב ראשון -מימוש AEC 5
שלב שני - מימוש Post-Filter 8
תוצאות 11
תוצאות במישור הזמן 12
תוצאות במישור הSTFT 12
סיכום חלק א 13
שינוי מספר הטפים 13
שינוי ה-$T60$ 14
חלק ב: שערוך הפרמטר- $T60$ 16
יצירת ה-database 16
בניית הרשת 21
פרמטרי הרשת 21
הכניסה לרשת 21
שכבות הרשת 22
מוצא הרשת 23
תוצאות 24
תוצאות עבור רשת הסיווג 24
תוצאות עבור רשת הרגרסיה 27
סיכום חלק ב 33
חלק ג: שילוב המערכות 34
תוצאות 35
בדיקת תקינות המערכת המשולבת - ערכים שרירותיים של $\boldsymbol{T60}$ 35
בדיקת ביצועי המערכת המשולבת-ערכים מרשת הסיווג של $\boldsymbol{T60}$ 37
בדיקת ביצועי המערכת המשולבת - ערכים מרשת הרגרסיה של $\boldsymbol{T60}$ 40
רעיונות להמשך 43
מסקנות וסיכום 43
ביבליוגרפיה 43

רקע לפרויקט

שימוש במכשירים מבוססי דיבורית בסביבה רועשת ומהדהדת זו בעיה נפוצה.
לדיבורית יש מיקרופון קטן, לפעמים מספר מיקרופונים- נתרכז בפרויקט זה במיקרופון אחד. במיקרופון זה מתקבל אות שמכיל את אות הדיבור הרצוי (אות הדובר הקרוב), רעשי רקע ואות הדהוד של הדובר הרחוק.
ההדהוד נוצר ממעבר של האות היוצא מהרמקול בחלל החדר ומגיע בחזרה למיקרופון. הפרעות אלה פוגעות בהבנת אות הדובר הקרוב. מערכת ביטול הד אקוסטית שואפת להשתמש במסנן מסתגל לביטול אות ההד שהתקבל. לאחר המעבר במערכת הממשת AEC יישאר הד שיורי. על מנת להדעיך את ההד השיורי נממש מסנן נוסף - Postfilter.

מימוש המערכות מבוסס על מודל הדהוד סטטיסטי. כאשר פרמטרי המודל תלויים בזמן ההדהוד של החדר. כלומר, מחדר לחדר זמן ההדהוד יכול להשתנות.
הערכת זמן ההדהוד בצורה עיוורת זו בעיה מאתגרת. ניתן להתגבר על אתגר זה באמצעות אלגוריתמים של למידה עמוקה. תחילה ננסח את הערכת זמן ההדהוד כבעיית סיווג , לאחר מכן ננסח כבעיית רגרסיה. עבור שתי הבעיות נשתמש ברשת קונבולוציה (CNN) משולבת עם Fully Connected.

בפרויקט זה ננסה לשלב בין המערכות AEC ו-Postfilter לבין השימוש באלגוריתמים של למידה עמוקה לשערוך הפרמטר $\mathbf{T_{60}}$ ובכך נגיע לתוצאות טובות יותר במערכת הכוללת ממה שיש כיום.

מטרת הפרויקט

בפרויקט זה נממש מערכת לזיהוי משך זמן הדהוד של חדר ע"י אלגוריתם למידה עמוקה – T_{60}. המערכת תתאמן על קטעי דיבור בעלי זמני הדהוד שונים על מנת ללמוד את הפרמטרים הפנימיים של הרשת. לאחר שנגיע לביצועים מספיק טובים הרשת תדע לסווג את זמן הדהוד לאותות דיבור הנקלטים בחדרים השונים בשגיאה שלא תשפיע על הביצועים בצורה ניכרת.

זמן ההדהוד המשוערך יוזן למערכת ביטול הד. מטרת מערכת ביטול ההד היא להדעיך עד כמה שניתן את ההד החוזר מהרמקול אל המיקרופונים כדי לאפשר קליטה נקייה של הדובר הרצוי. כדי להדעיך את ההד באופן מיטבי משתמשים באות הרפרנס הקיים במערכת ומשערכים את המערכת האקוסטית מהרמקול אל המיקרופון. ידיעת זמן ההדהוד של החדר תעזור בקביעת האורך של המערכת האקוסטית. לבסוף, נשתמש במסנן נוסף כדי להדעיך הד שיורי. המסנן הנוסף זקוק גם הוא לשערוך של זמן ההדהוד של החדר.

התוצר הסופי יהיה מערכת כוללת אשר מכילה:

1. מערכת למידה עמוקה של זמן הדהוד.
2. מערכת ביטול אקו המשתמשת בזמן ההדהוד לקבלת ביטול הד מיטבי.

הערות:

T_{60} – הזמן שלוקח להדהוד לרדת ב-$60[dB]$.
הפרמטר תלוי בפרמטרים שונים של הבעיה, מיקום המיקרופון, מיקום הדובר, גודל החדר ועוד.
הסיבה שמודדים דווקא ירידה של $60[dB]$ ולא הנחתה אחרת היא מכיוון שלאחר הנחתה כזאת האוזן האנושית לא יכולה לשמוע. כלומר, כבר לא נוכל להבחין ברעש שנמוך מ-$60[dB]$.

בשלב הראשון מימשנו את מערכת ביטול ההד בעזרת אלגוריתמים של עיבוד אותות. בשלב זה, הנחנו שזמן ההדהוד ידוע. לאחר מכן, שערכנו את זמן ההדהוד בעזרת למידה עמוקה. לבסוף, שילבנו בין המערכות.

חלק א: מימוש המסננים לביטול ההד כאשר זמן ההדהוד ידוע

תרשים כולל של המערכת:

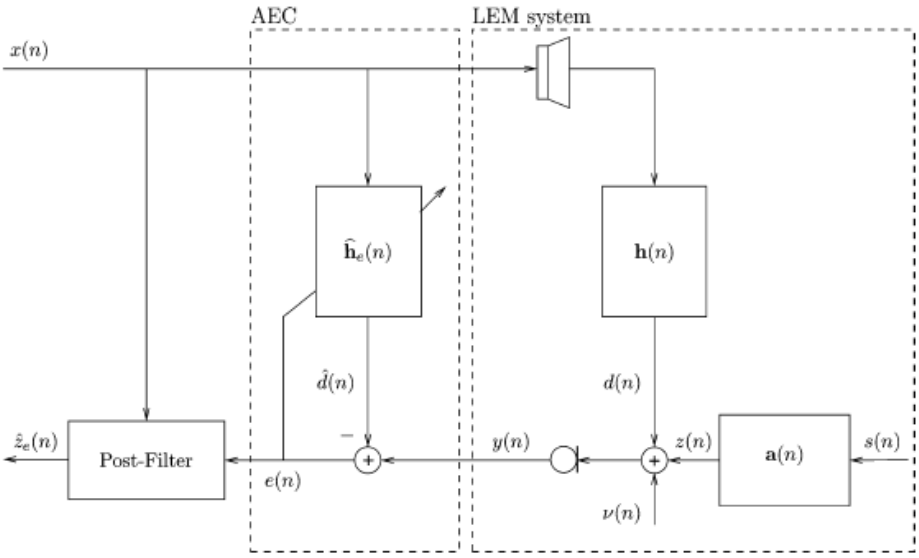
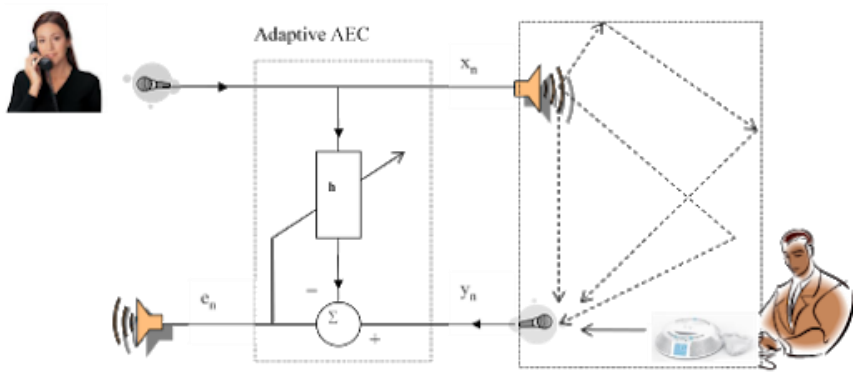


Fig. 1. Acoustic echo canceller with postfilter.

- $x[n]$ - אות הדובר הרחוק.
- $s[n]$ - אות הדובר הקרוב (הדיבור הרצוי).
- $h[n]$ - מערכת אקוסטית (החדר) מהרמקול למיקרופון.
- $y[n]$ - סיגנל שמתקבל במיקרופון.
- $z[n]$ - דיבור מהדהד, הדובר הקרוב שמגיע למיקרופון.
- $d[n]$- הד אקוסטי, הדובר הרחוק שמגיע למיקרופון.
- $v[n]$ - רעש סביבתי.
- $a[n]$- AIR (acoustic impulse response) מערכת אקוסטית מפה הדובר למיקרופון.

שלב ראשון -מימוש AEC



האלגוריתם מקבל בקלט את האות $y[n]$ הכולל את הדיבור הרחוק והקרוב אחרי המעבר שלהם במערכת האקוסטית וכולל את הרעש הסביבתי. בנוסף, האלגוריתם מקבל בקלט את $x[n]$, סיגנל הרפרנס.
האלגוריתם מנסה לשערך את $h[n]$ - המערכת האקוסטית של החדר מהרמקול אל המיקרופון.
לאחר שערוך המערכת האקוסטית של החדר - $\hat{h}_e[n]$ נעביר את סיגנל הרפרנס במערכת ונקבל את - $\hat{d}[n]$.
נחסר את הסיגנל מהאות הכולל שיוצא מהרמקול. לבסוף, נקבל במוצא את האות $e[n]$ שזה אות הדובר הקרוב עם הרעש הסביבתי.
מכיוון ששערוך המערכת האקוסטית לא מדויק יהיה לנו הד שיורי. את ההד הזה נרצה להוריד בעזרת מסנן נוסף- Postfilter. המסנן הנוסף דואג להדעכת ההד השיורי של הדובר הרחוק, הד הדובר הקרוב ורעשי רקע.

שלבי האלגוריתם -AEC (Acoustic Echo Canceller):

- נשערך את $h[n]$ - המערכת האקוסטית של החדר מהרמקול למיקרופון. נסמן את השיערוך $\hat{h}_e[n]$.
- נבצע קונבולוציה בין סיגנל הרפרנס למסנן המשוערך: $\hat{d}[n] = x[n] * \hat{h}_e[n]$.
- נפחית את התוצאה מ-$y[n]$: $e[n] = y[n] - \hat{d}[n]$.

בעיית שערוך זו היא בעיה מוכרת, יש מספר אלגוריתמים שניתן להשתמש בהם כדי לשערך את המערכת האקוסטית, במאמר השתמשו באלגוריתם NLMS. אנחנו השתמשנו באלגוריתם RLS מהסיבה שאלגוריתם זה מתכנס מהר יותר מהאלגוריתמים האחרים.

הערה:

המערכת האקוסטית שאנחנו משערכים- $\hat{h}_e[n]$ בעלת אורך סופי N_e שתלוי ב-T_{60} . N_e קטן יותר מאורך המסנן המקורי- N_h.

מימוש האלגוריתם-RLS:

בפיתוח האלגוריתם אנחנו מניחים שאין מידע סטטיסטי על הפרמטרים או המדידות, אך ידוע מודל המקשר ביניהם:

$$d[n] = \sum_{j=0}^{N_e-1} h_{e,j}[n]x[n-j]$$

בכתיב וקטורי ניתן לכתוב:

$$d_n = h_n^T x_n$$

כאשר x_n הינו וקטור באורך N_e שמשתנה בכל נקודת זמן.

x_n - אות הכניסה , נרצה למצוא את המסנן כך שמוצא המערכת יהיה קרוב ככל האפשר לאות הרצוי - d_n.

כדי למצוא את מקדמי המסנן האופטימליים נגדיר פונקציית מטרה - שגיאה ריבועית ממוצעת.
פונקציית המטרה:

$$e_n = (y_n - h_n^T x_n)^2$$

המטרה שלנו היא למזער את פונקציית המטרה וכך למצוא את המסנן האופטימלי:

$$\hat{h}_n = argmin_{h_n}(y_n - h_n^T x_n)^2$$

נגזור את הביטוי ונשווה לאפס:

$$\frac{de_n}{dh_n} = 2(y_n - h_n^T x_n)x_n^T = 0$$

$$\rightarrow \hat{h}_n = y_n x_n^T (x_n x_n^T)^{-1}$$

כדי שהאלגוריתם שלנו יהיה מסתגל, כלומר, נרצה שלכל דגימה חדשה נעדכן את המסנן ולא נחשב מחדש, נגדיר:

$$A_n = \alpha A_{n-1} + (1-\alpha) y_n x_n^T$$

$$B_n = \alpha B_{n-1} + (1-\alpha) x_n x_n^T$$

נקבל:

$$\hat{h}_n = A_n B_n^{-1}$$

מימוש בתחום התדר

תחילה מימשנו את האלגוריתם בתחום הזמן.
במצב זה עברנו על כל אורך סיגנל הרפרנס בלולאה כדי לשערך את המסנן המסתגל בכל נקודת זמן.
כאשר את המסנן יצרנו ע"י rir_generator. בחרנו שאורך המסנן יהיה 512 דגימות לכן המטריצה שלנו הייתה בגודל זה.
קיבלנו כי האלגוריתם צריך להפוך מטריצה בגודל 512×512 עבור כל דגימה חדשה!
לכן כדי לייעל את כמות החישובים עברנו למישור זמן-תדר, מישור ה-STFT.
בחרנו את מסנן האנליזה להיות $hamming$ באורך 512 עם חפיפה של 75%. לכן כדי לבסות את אותו התחום במישור ה-STFT נצטרך בכל פעם לקחת 4 פריימים של הזמן. לכן בכל איטרציה בלולאה נהפוך מטריצה בגודל 4×4 בלבד.
במקרה זה נוסף לנו מימד נוסף - מימד התדר לכן עבור כל נקודת זמן הרצנו לולאה פנימית לכל התדרים.

כפי שלמדנו, במקרה שאנחנו במישור התדר - במקום סינון לקבלת כל איבר במוצא נבצע מכפלה - חיסכון נוסף שהתקבל בזמן הריצה.

הקוד שמימשנו:

```
for l=1:sizeX(1) % run along the time;
    waitbar(l/sizeX(1),h);

    for k=1:win_len % run along the frequency
        if l<L_max
            alpha=0.99;
        else
            alpha=0.999;
        end

        X_buffer(:,k) = [X(l,k); X_buffer(1:end-1,k)];

        %%% Cross correlation update
        A(:,k) = alpha.*A(:,k)+(1-alpha)*Y(l,k)'*X_buffer(:,k);

        %%% Auto correaltion update using Matrix inverion lemma
        B(:,:,k) = alpha.*B(:,:,k) + (1-alpha) * X_buffer(:,k) * X_buffer(:,k)';
        H(:,k) = (A(:,k)'/( B(:,:,k) + eye(T) * ( trace(B(:,:,k)) * 0.01 + 10^-5)  ))';

        %%% echo estimation
        D_hat(l,k) = H(:,k)'*X_buffer(:,k);

        %%% echo cancellation
        E(l,k)= Y(l,k)-D_hat(l,k);
```

כאשר:

l - מספר הפריים בתחום הזמן

k - התדר

רצנו על כל התדרים בכל פריים של הזמן וכך מצאנו לכל נקודת זמן תדר שונה את מוצא האלגוריתם שפי שהסברנו קודם.

* כדי למנוע בעיות בהיפוך המטריצה הוספנו ערכים קטנים לאלכסון, יכול לפגוע מעט בביצועים אבל, זה עדיף מאשר בעיות סינגולריות.

* כדי שהלמידה תיהיה רק באזור בו הדובר הרחוק משמעותי שינינו את הפרמטר α בהתאם לנוכחות הדוברים בסיגנל שבחרנו. כאשר L_{max} מגדיר את הפריים האחרון בו רק הדובר הרחוק מדבר.

המוצא שהתקבל:

$$E(l,k) = Z_e(l,k) + Z_r(l,k) + E_r(l,k) + V(l,k)$$

בשלב זה המוצא מכיל עדיין הד שיורי - $e_r[n]$ אותו ננסה להוריד בשלב הבא - שלב ה-Postfilter.

הערה: במאמר חילקו את $z[n]$, הדובר הקרוב, ל-$z_e[n]$ - הגעות ראשוניות של הדובר הקרוב ול-$z_r[n]$ – הגעות מאוחרות של הדובר הקרוב. אנחנו נתייחס לשניהם כאות יחיד.

במוצא ה-AEC במישור הזמן מתקבל:

$$e[n] = z[n] + e_r[n] + v[n]$$

נסמן: $\hat{v}[n] = e_r[n] + v[n]$

ונקבל: $e[n] = z[n] + \hat{v}[n]$

שלב שני - מימוש Post-Filter

אנחנו מעוניינים לשערך את $z[n]$. לשם כך נשתמש במסנן ווינר לסינון הרעש:

$$\hat{z}[n] = w[n] * e[n]$$

נעשה את הניתוח למציאת המסנן האופטימלי במישור התדר, נרצה למזער את השגיאה הריבועית הממוצעת:

$$\begin{aligned}\widehat{W}(l,k) &= argmin\left\{E\left(\left|\hat{Z}(l,k) - Z(l,k)\right|^2\right)\right\} \\ &= argmin_{W(l,k)}\left\{E\left(|W(l,k)\cdot E(l,k) - Z(l,k)|^2\right)\right\} \\ &= argmin_{W(l,k)}\left\{E\left(\left|W(l,k)\cdot\left(Z(l,k) + \hat{V}(l,k)\right) - Z(l,k)\right|^2\right)\right\} \\ &= argmin_{W(l,k)}\left\{E\left(\left|(W(l,k)-1)Z(l,k) + W(l,k)\cdot\hat{V}(l,k)\right|^2\right)\right\}\end{aligned}$$

$$\underbrace{=}_{Z,\hat{V}\text{ חסרי קורלציה}} argmin_{W(l,k)}\left\{(W(l,k)-1)^2E\{Z^2(l,k)\} + W^2(l,k)\cdot E\{\hat{V}^2(l,k)\}\right\}$$

$$\rightarrow \widehat{W}(l,k) = argmin_{W(l,k)}\left\{(W(l,k)-1)^2E\{Z^2(l,k)\} + W^2(l,k)\cdot E\{\hat{V}^2(l,k)\}\right\}$$

כדי למצוא את המינימום נגזור לפי $W(l,k)$ ונשווה לאפס:

$$2(W(l,k)-1)E\{Z^2(l,k)\} + 2W(l,k)\cdot E\{\hat{V}^2(l,k)\} = 0$$

$$W(l,k)\left(E\{Z^2(l,k)\} + E\left\{\hat{V}^2(l,k)\right\}\right) = E\{Z^2(l,k)\}$$

$$\rightarrow W(l,k) = \frac{E\{Z^2(l,k)\}}{E\{Z^2(l,k)\} + E\left\{\hat{V}^2(l,k)\right\}} = \frac{\frac{E\{Z^2(l,k)\}}{E\left\{\hat{V}^2(l,k)\right\}}}{\frac{E\{Z^2(l,k)\}}{E\left\{\hat{V}^2(l,k)\right\}} + 1} = \frac{SIR}{SIR+1}$$

כאשר:

$$SIR \equiv \frac{E\{Z^2(l,k)\}}{E\left\{\hat{V}^2(l,k)\right\}}$$

נגדיר הספק בצורה הבאה:

$$E\{Z^2(l,k)\} \equiv \lambda_z(l,k)$$

$$E\left\{\hat{V}^2(l,k)\right\} \equiv \lambda_{e_r}(l,k) + \lambda_v(l,k)$$

כאשר הפרדנו את הרעש הכולל של מוצא ה-*AEC* לשני חלקים - הד שיורי ורעש סביבתי. כמו שציינו, אנחנו מניחים ש-$\lambda_{e_r}(l,k)$ מכיל את כל ההד השיורי, גם של הדובר הקרוב וגם של הדובר הרחוק.
בסה"כ נקבל:

$$\frac{E\{Z^2(l,k)\}}{E\left\{\hat{V}^2(l,k)\right\}} = \frac{\lambda_z(l,k)}{\lambda_{e_r}(l,k) + \lambda_v(l,k)} \equiv \xi(l,k) \equiv a\ priori\ SNR$$

יחס האות לרעש האפריורי.

לכן נקבל שמסנן ווינר הוא:

$$G_{H1}(l,k) = \frac{\xi(l,k)}{\xi(l,k)+1}$$

נציין שזה מסנן ווינר שמתקבל כאשר יש לנו אות רצוי, במקרה שאין לנו אות רצוי (רק הד ורעש) נקבל מסנן שונה- $G_{H0}(l,k)$, מסנן זה דואג שכאשר אין דובר רצוי תהיה הנחתה שונה וגדולה יותר. במקרה זה נרצה שעוצמת ההד תהיה נמוכה מעוצמת הרעש. כך נקבל שיפור בביצועים של המסנן. מכיוון שאנחנו בפרויקט לא הוספנו גלאי סף שאומר לנו האם יש דיבור קרוב או שאין לא הוספנו את המסנן השני.
באופן כללי, מה שנצטרך לעשות זה להפעיל כל פעם את אחד המסננים לפי גלאי הסף ההסתברותי.

כעת נצטרך לשערך את $\xi(l,k)$:

תחילה נגדיר גודל נוסף, יחס האות לרעש האפוסטריורי:

$$\gamma(l,k) = \frac{\lambda_e(l,k)}{\lambda_{e_r}(l,k)+\epsilon} \equiv a\ posetriori\ SNR$$

כאשר כמו שהגדרנו קודם:

$$\lambda_e(l,k) = E\{E^2(l,k)\}$$

$$\lambda_{e_r}(l,k) = E\{E_r^2(l,k)\}$$

כדי לשערך את ה-*SNR* בצורה חלקה נבצע החלקה עם ה-*SNR* בנקודת הזמן הקודמת:

$$\hat{\xi}(l,k) = \eta \cdot \gamma_{pre}(l-1,k) + (1-\eta)\cdot \max(\psi(l,k),0)$$

כאשר נגדיר:

$$\gamma_{pre}(l,k) = \gamma(l,k)\cdot G_{H1}(l,k)^2$$

$\gamma_{pre}(l,k)$- הספק מוצא המסנן ברגע הקודם חלקי הספק הרעש, כלומר ה-*SNR* האפריורי ברגע הקודם:

$$\gamma_{pre}(l,k) = \gamma(l,k)\cdot G_{H1}(l,k)^2 = \frac{\lambda_e(l,k)}{\lambda_{e_r}(l,k)+\epsilon}\cdot G_{H1}(l,k)^2 = \frac{E\{E^2(l,k)\cdot G_{H1}(l,k)^2\}}{\lambda_{e_r}(l,k)+\epsilon}$$
$$= \frac{E\{E_{PF}(l,k)^2\}}{\lambda_{e_r}(l,k)+\epsilon} = \frac{E\{\hat{Z}_e(l,k)^2\}}{\lambda_{e_r}(l,k)+\epsilon}$$

במוצא המסנן יש לנו את האות הנקי לכן בסה"כ מה שנקבל זה את ה-*SNR* ברגע הקודם.

נוסיף את ההשפעה של הכניסה הנוכחית:

$$\psi(l,k) = \gamma(l,k) - 1 = \frac{\lambda_e(l,k)}{\lambda_{e_r}(l,k)+\epsilon} - 1 = \frac{\lambda_e(l,k) - \lambda_{e_r}(l,k) + \epsilon}{\lambda_{e_r}(l,k)+\epsilon} \cong SNR$$

לכן ההחלקה שאנחנו עושים היא בין ה-*SNR* הקודם ל-*SNR* הנוכחי. אנחנו נותנים משקל גדול יותר לזמן הקודם כדי שלא יהיו שינויים חדים במסנן והוא יהיה חלק יותר. שערוך כזה נקרא *decision directed*.
בחרנו: $\eta = 0.95$.
שיערוך הספק מוצא ה-*AEC*:

$$\lambda_e(l,k) = \begin{cases} \eta_e \cdot \lambda_e(l-1,k) + (1-\eta_e)\cdot |E(l,k)|^2, & |E(l,k)|^2 < \lambda_e(l-1,k) \\ 0.5\cdot \lambda_e(l-1,k) + 0.5\cdot |E(l,k)|^2\ , & else \end{cases}$$

גם כאן ביצענו החלקה, אבל, כאשר הספק המוצא נמוך יותר מהשערוך הקודם (כנראה שיש רק רעש, לא נרצה להתחשב במוצא הזה בצורה משמעותית) ניתן משקל גדול יותר לרגע הקודם, אחרת - נתחשב בשני הגדלים בצורה שווה.

בחרנו: $\eta_e = 0.95$.

שערוך הספק ההד השיורי:

$$\lambda_{e_r}(l,k) = \alpha(k) \cdot \left(\lambda_{e_r}(l-1,k) + c(l,k) \cdot \lambda_x\left(l - \frac{N_e}{R}, k\right)\right)$$

ההחלקה במקרה זה מתבצעת לפי ההד השיורי בזמן הקודם והתחשבות באות הדובר הרחוק שגם נתון לנו.

אנחנו מחפשים את השערוך של ההד. לכן, אנחנו משערכים את פונקציית התמסורת של תגובת החדר שלנו - $H(l,k)$ על ידי החלקה על מספר פריימים שונים ומסננים את הדובר הרחוק במסנן המשוערך. מכיוון שאנחנו מנסים לשערך את ההד השייורי ניקח את נקודת הזמן הנוכחית ונחסר ממנה $\frac{N_e}{R}$. כאשר $\frac{N_e}{R}$ זה מספר הפריימים שנחזור אחורה, בהתאם לאורך המסנן - שקובע את זמן ההדהוד כדי לשערך את ההד שהגיע בנקודת הזמן הנוכחית למוצא ה-*AEC*.

נחשב את הספק אות הדיבור הרחוק בצורה דומה להספק מוצא ה-*AEC*:

$$\lambda_x(l,k) = \eta_x \cdot \lambda_x(l-1,k) + (1-\eta_x)|X(l,k)|^2$$

כאשר: $\eta_x = 0.7$, יש משקל גדול יותר לערך הקודם.

$$\alpha(k) = e^{-\frac{2\rho(k)R}{f_s}}$$

גורם הדעיכה של ההדהוד בין שני זמנים עוקבים, תלוי בזמן ההדהוד של החדר - $T_{60}(k)$. עבור כל תדר הדעיכה של הקול שונה לכן גורם הדעיכה תלוי ב-*k*. כדי לפשט אנחנו הנחנו שזמן ההדהוד קבוע בכל התדרים.

$$\rho(k) = \frac{3 \cdot \ln(10)}{T_{60}(k)}$$

$$c(l,k) = 0.95 \cdot c(l-1,k) + 0.05 \cdot \sum_{i=-w}^{w} |H(l,k+i)|^2 b(i)$$

התפקיד של $c(l,k)$ זה שערוך תגובת החדר בכל נקודת זמן, מכיוון שאנחנו מניחים שזמן ההדהוד בחדר עשוי להשתנות בכל נקודת זמן נבצע החלקה בין זמנים שונים.
למסנן המשוערך (ממוצא ה*AEC*) נבצע החלקה על מספר פסי תדר עם חלון *hamming* כדי שהתוצאה תהיה חלקה יותר גם מבחינת התדרים השונים.

באשר:

$$\sum_{i=-w}^{w} b(i) = \sum_{i=-w}^{w} \frac{W_{hamming}(i)}{\sum_{j=-w}^{w} W_{hamming}(j)} = 1$$

חלון *hamming* מנורמל.

לבסוף מוצא המערכת יהיה:

$$\mathrm{E_{PF}} = \max(G_{H1}(l,k), 0.1) \cdot E(l,k)$$

נדאג תמיד להשאיר רעש במוצא ולא לאפס את המוצא לחלוטין. כאשר המסנן בעל אמפליטודה נמוכה לא נרצה להנחית לגמרי את האות במוצא, נרצה שיישאר קצת רעש. לכן נדאג תמיד שהמסנן ישאיר לפחות 10% מהאות במוצא ה*AEC*.

תוצאות

בדקנו את מוצא המסננים כאשר ידוע לנו זמן ההדהוד של החדר.
קבענו בשלב זה שמספר הפריימים בכל שלב יהיה קבוע ושווה ל-4, מבחינת חומרה קשה לממש היפוך מטריצות גדולות יותר לכן קבענו את ערך זה.

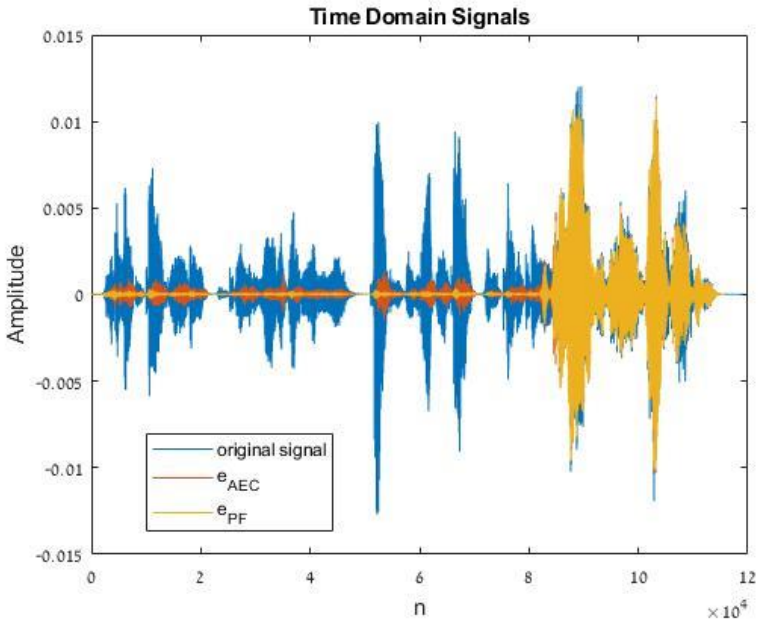
כדי לייצור את הדובר הרחוק והדובר הקורב לקחנו שני דוברים שונים מהמאגר *TIMIT* כאשר לכל דובר שרשנו כמה קטעי קול כדי להגיע להקלטה באורך הרצוי. יצרנו הקלטה באורך של כ-7 שניות, כאשר ב-$5[s]$ הראשונות רק הדובר הרחוק מדבר וב-$2[s]$ האחרונות שני הדוברים מדברים יחד.

בנינו מסננים בעזרת *rir- generator* כאשר בחרנו שזמן ההדהוד בחדר יהיה: $\beta = 0.2[s]$. כאשר, β מסמן את ה-T_{60} האמיתי של החדר. ביצענו קונבולוציה בין המערכות לדוברים. לאחר מכן, חיברנו ביניהם והוספנו רעש של $30[dB]$.

```
%% INPUT
%Signals
[male1, fs] = audioread('SAlmale.wav');
male2 = audioread('SAlmale3.wav');
female = audioread('SAlfemale3.wav');

x = [male1',male2'];
s = [zeros(1,length(male)-length(female')),female'];

load('h_n02.mat'); h=h_n02;
len_h = length(h);
load("a_n02.mat"); a=a_n02;
len_a = length(a);
%Convolution
z = conv(s,a);
d = conv(x,h);
%Noise
v = rand(1,length(d));
dVSv = sqrt(sum((d).^2)./(sum((v).^2)*10^3));
v = dVSv*v;
dVSz = sqrt(sum((d).^2)./(sum((z).^2)));
z = dVSz*z;

y = z+v+d;

%%%%%%%%%%%%%%%%%%%%%% DELAY
cross=xcorr(x,y);
[~, p]=max(cross);
delay_AEC = (length(cross)-1)/2-p;
x =[zeros(1,delay_AEC) x(1:end-delay_AEC)];
```

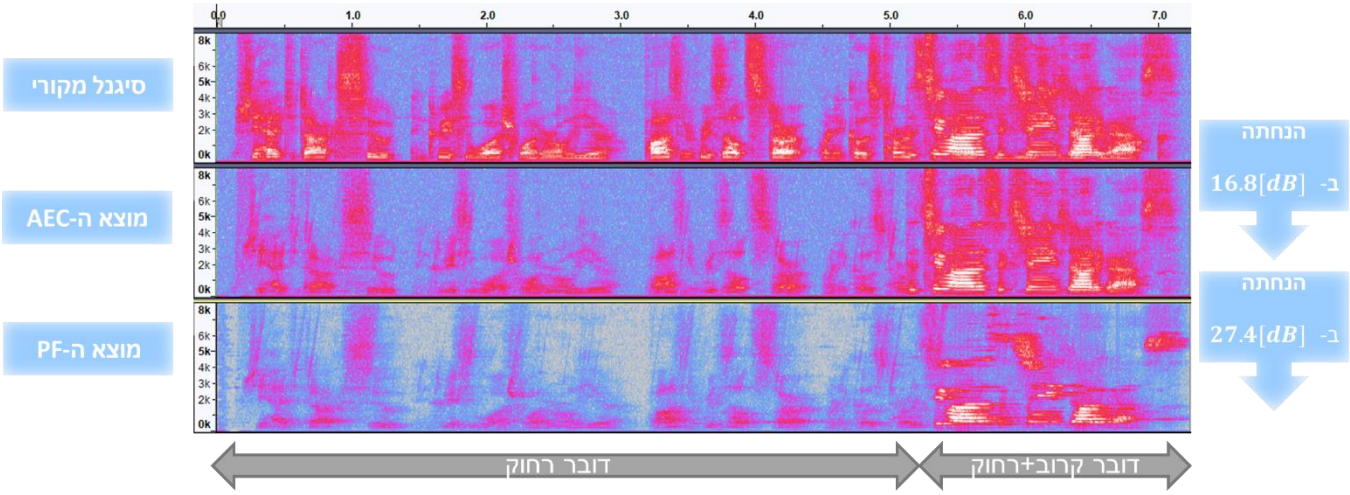
נציין כי האלגוריתם לומד את מקדמי המסנן רק כאשר הדובר הרחוק משמעותי, לכן בחרנו שעיקר הלמידה של המסנן יהיה בזמן שבו הדובר הזה מדבר. לכן, נבצע את עיקר הלמידה על ה-5 שניות הראשונות.

תוצאות במישור הזמן



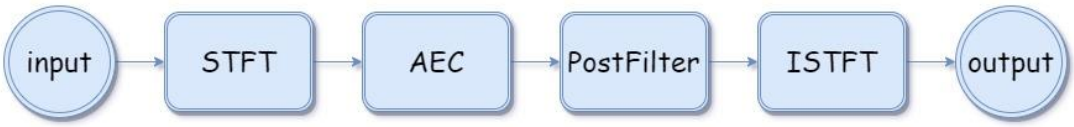
ניתן לראות שאכן האמפליטודה של האות יורדת באזור של הלמידה בצורה משמעותית. ולאחר המעבר במסנן יש הנחתה נוספת של האות מה-*postfilter*. באזור בו הדובר הקרוב מדבר אין הנחתה כפי שרצינו.

תוצאות במישור הSTFT



סיכום חלק א

תרשים המערכת:



בדקנו איך משפעים שני פרמטרים שונים על ביצועי המערכת כתלות בחדרים עם רמות הדהוד שונות:

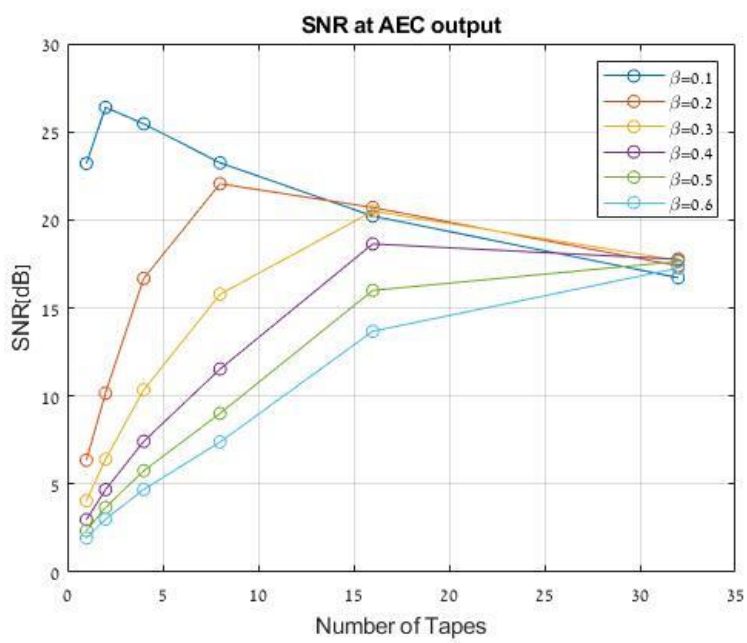
- מספר הטפים (האורך של המערכת האקוסטית כשמשערכים במישור ה-STFT).
- ה-T_{60} שנכניס לאלגוריתם.

לקחנו שני דוברים שונים ממאגר *TIMIT* ושרשרנו מספר קטעי קול שלהם לאורך כולל של 50 שניות. כאשר ב-$35[s]$ הראשונות רק הדובר הרחוק מדבר וב-$15[s]$ האחרונות שני הדוברים מדברים יחד. בנינו מסננים בעזרת *rir- generator* כאשר דאגנו שיהיו לנו את זמני ההדהוד הבאים: $[0.1: 0.1: 0.6][s]$.
בכל הרצה דאגנו לשנות רק פרמטר אחד כדי שנוכל לדעת מהי ההשפעה של כל אחד מהפרמטרים.

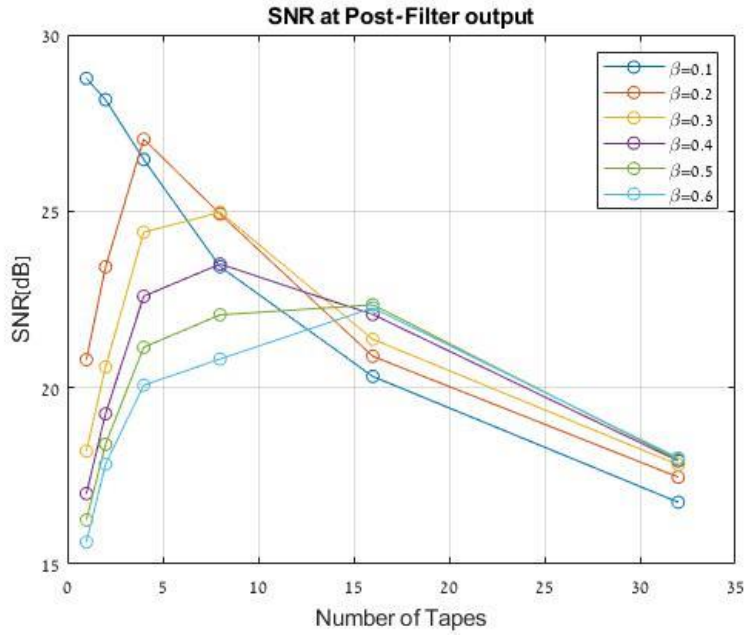
שינוי מספר הטפים

לבדיקת השפעת הפרמטר שינינו את מספר הטפים שהזנו לאלגוריתם של ה-AEC .
הכנסנו ערכים שונים בקפיצות של חזקות 2 (יעיל יותר מבחינת חומרתית). עבור ה-PF השתמשנו בערך האמיתי של זמן ההדהוד של החדר, כלומר $\beta = T_{60}$.
התוצאות שהתקבלו:

מוצא הAEC:



מוצא הPost-Filter:



לפי התוצאות שקיבלנו ניתן להסיק:

<u>עבור ה-AEC:</u>

- עבור מספר הטפים הנכון מתקבלות התוצאות הטובות ביותר.
- בהדהוד נמוך - ככל שמגדילים את מספר הטפים מעבר למספר האמיתי התוצאות נפגעות.

← חשוב לדעת את מספר הטפים במדויק.

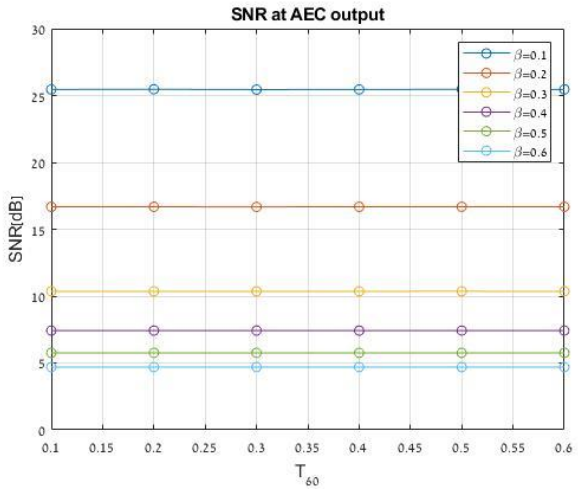
<u>עבור ה-PF:</u>

- ה-PF לא נפגע כתוצאה מהגדלה במספר הטפים. בהדהוד נמוך בגלל שמוצא ה-AEC נפגע אז גם במוצא שלו התוצאה נפגעת.

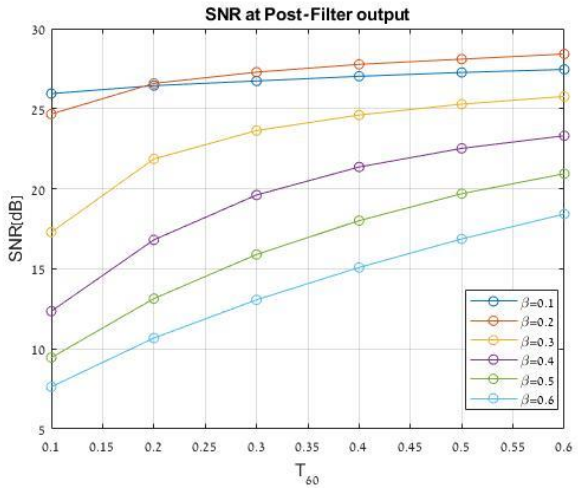
שינוי ה-T_{60}

כדי לבדוק את ההשפעה של הפרמטר השני, הזנו ל-AEC מספר טפים קבוע ושווה ל- 4. הסיבה שבחרנו 4 היא מסיבות של חומרה. (בד"כ מערכות לא יכולות להפוך מטריצות גדולות יותר מ-4x4 ולעבוד בצורה טובה בזמן אמת).
בכל הרצה שינינו את ה-T_{60} שאנחנו מזינים ל-PF.
התוצאות שהתקבלו:

מוצא הAEC:



מוצא הPost-Filter:



לפי התוצאות שקיבלנו ניתן להסיק:

<u>עבור ה-AEC:</u>

- כפי שציפינו, התוצאות לא משתנות כתלות בפרמטר T_{60}.

<u>עבור ה-PF:</u>

- עבור הדהוד נמוך – אין כ"כ השפעה של הפרמטר T_{60}. כלומר, אם טועים ונותנים הדהוד גבוה יותר מהדהוד האמיתי התוצאות לא נפגעות שכן אין הדהוד נוסף שאפשר להוריד.
- עבור הדהוד גבוה – יש השפעה של הפרמטר T_{60}. אם טועים ונותנים הדהוד נמוך אנחנו לא מצליחים להוריד את כל ההדהוד.

← ידיעת הפרמטר בצורה מדויקת תשפר את ביצועי המערכת.

ניתן לראות משתי התוצאות האחרונות את החשיבות של ידיעת זמן ההדהוד של החדר עבור שתי המערכות ה-AEC וה-PF כדי להגיע לביצועים מיטיבים.

חלק ב: שערוך הפרמטר- T_{60}

כפי שציינו בהצגת הבעיה זמן הדהוד של החדר משתנה מחדר לחדר ותלוי בפרמטרים שונים כמו מיקום המיקרופון, מיקום הרמקול, גודל החדר וכו'. דבר זה מקשה להעריך בצורה עיוורת את זמן ההדהוד. לכן כדי להעריך את זמן ההדהוד עבור כל הקלטה נשתמש בכלים של למידה עמוקה.

השתמשנו בשפת python ובחבילה בשם PyTorch ובנינו רשת CNN משולבת עם Fully Connected. הרשת מקבלת תמונת זמן תדר-STFT שמייצגת את ההקלטה שמתקבלת בחדר עם ההדהוד ומחזירה את זמן ההדהוד של החדר-T_{60}.

לצורך בניית ה-dataset לקחנו קטעי קול נקיים (הוקלטו בחדר אקוסטי) וביצענו קונבולוציה בינם לבין תגובות התדר של חדרים שונים. את החדרים יצרנו בעזרת שימוש ב-rir generator.

חילקנו את ההקלטה למקטעים באורך 4 שניות כאשר בין המקטעים יצרנו חפיפה של חצי שניה. לכל מקטע ביצענו התמרת STFT כפי שנפרט בהמשך. בשלב אימון הרשת הכנסנו למערכת את תמונת ה-STFT לאחר שעברה עיבוד מקדים ובעזרתה אימנו את הרשת ובדקנו את הביצועים.

במוצא של הרשת מתקבל זמן ההדהוד של החדר. במהלך הפרויקט בנינו שתי רשתות קונבולוציה שהמוצא שלהם שונה:

1. רשת סיווג - מחזירה class שמייצג את זמן ההדהוד שהכי קרוב לזמן האמיתי. זמן ההדהוד של כל אחד מתוך חמשת ה-class-ים היה: $0.2,0.3,0.4,0.5,0.6[s]$. כדי לאמן את הרשת הזו השתמשנו ב-cross entropy loss.
2. רשת רגרסיה – מחזירה מספר רציונאלי שמייצג את זמן ההדהוד של החדר. כדי לאמן את הרשת הזו השתמשנו ב-*MAE-loss* מנורמל (*MAE-Mean Absolute Error*).

יצירת ה-database

על מנת שהרשת תוכל לפענח את זמן ההדהוד של החדר נצטרך לאמן אותה בעזרת דוגמאות מתויגות. כלומר, דוגמאות שזמן ההדהוד שלהם ידוע.

על מנת להבטיח שהמודל שאימנו לא יגיע ל-overfitting או יתמקד בדברים שנובעים מתהליך סינתזת התמונות, יצרנו database שמכיל דוגמאות נפרדות לשלב הלמידה ולשלב הבדיקה (test).

Setup טיפוסי זה לחלק את הדוגמאות לשלושה חלקים נפרדים: סט אימון כדי לאמן את הרשת, סט ולידציה (validation) כדי לוודא שהרשת לא מגיעה ל-overfitting במהלך האימון וסט בדיקה כדי לבדוק את רמת הדיוק והביצועים של הרשת. פירוט של כמות הדוגמאות:

Type	Samples
Training	30,000
Validation	10,000
Testing	5,000

כדי לייצר את הדוגמאות השתמשנו בקטעי דיבור נקיים ללא מעבר בשום מערכת. עבור כל דוגמא הגרלנו דובר מ-TIMIT CD ושרשנו הקלטות שונות של אותו דובר (שגם נבחרו בצורה רנדומלית) עד שהגענו לזמן כולל של בערך 20 שניות.

```
x =[];
folder_num = randi(8);
your_folder = ['TIMIT CD\TIMIT\TRAIN\DR',num2str(folder_num)];
%d=pwd()
files = dir(your_folder);
directoryNames = {files([files.isdir]).name};
directoryNames=directoryNames(3:end);
n=numel(directoryNames);
dic = {dir([your_folder,'\',char(directoryNames(n)), '\*.wav']).name};
fs=16000;
while (length(x)<20*fs) %20[s]
    idx=randi(numel(dic));
    file=dic{idx};
    [signal,fs] = audioread([your_folder,'\',char(directoryNames(n)),'\',file]);
    if size(signal,1)<size(signal,2),signal=signal.';end
    x = [x ; signal];
end
```

בנוסף, הגרלנו מערכת ע"י שימוש ב-rir generator:

```
%% room - h
[h,information] = createH();
```

אלגוריתם זה מקבל כבניסה את מימדי החדר, את מיקום הדוברים והמיקרופונים, את זמן ההדהוד ועוד נתונים ויוצר מערכת.

הקוד המלא:

```
function [h_n,information] = createH()

    load handel.mat

    x = 3 + 3*rand(1);              % x position (m) - number in the range
[3,6](m)
    y = 3 + 3*rand(1);              % y position (m) - number in the range
[3,6](m)
    L = [x y 3];                    % Room dimensions [x y z] (m)
    c_k = 340;                      % Sound velocity (m/s)
    fs = 16000;                     % Sample frequency (samples/s)
    Vbeta = (0.15:0.005:0.65);      % Possible values for beta
    idx = randi(numel(Vbeta));      % Random index selection
    beta = Vbeta(idx);              % Reverberation time (s)
    n = fs*beta;                    % Number of samples
    mtype = 'omnidirectional';      % Type of microphone
    order = -1;                     % -1 equals maximum reflection order!
    dim = 3;                        % Room dimension
    orientation = 0;                % Microphone orientation (rad)
    hp_filter = 1;                  % Disable high-pass filter

    x_position = [x*rand(1) y*rand(1) 1];  % Source position [x y z] (m)

    distance= 0.3+1.7*rand(1);      % number in the range [0.3,2](m)
    mic_x_temp = distance*rand(1);  % Random x position
    mic_y_temp = sqrt(distance^2-mic_x_temp^2); % y^2+x^2 = distance^2
```

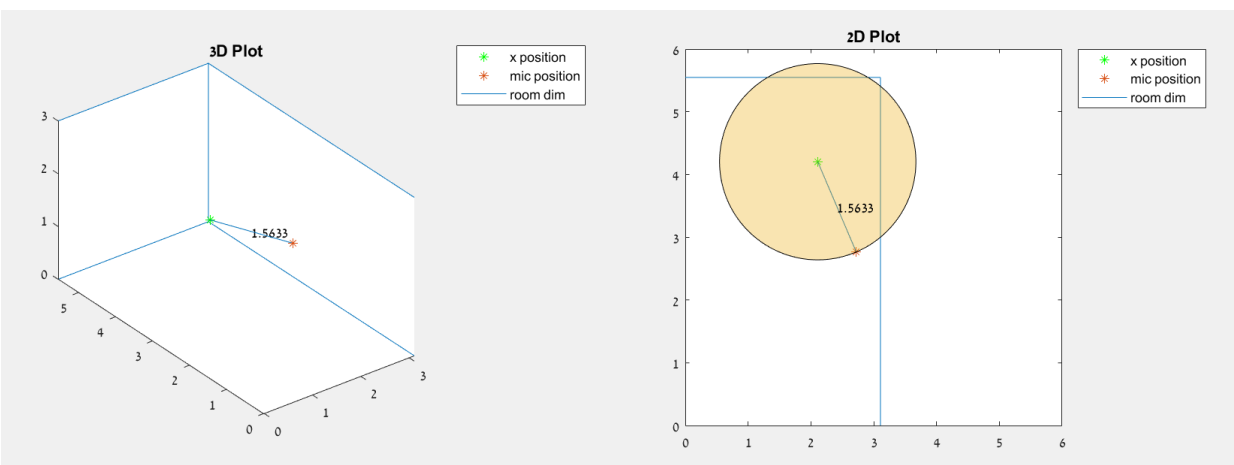
```
    if x_position(1)+mic_x_temp >= x
        mic_x=x_position(1)-mic_x_temp;
    else
        mic_x=x_position(1)+mic_x_temp;
    end

    if x_position(2)+mic_y_temp >= y
        mic_y=x_position(2)-mic_y_temp;
    else
        mic_y=x_position(2)+mic_y_temp;
    end

    mic_position = [mic_x mic_y 1];         % Receiver position [x y z] (m)

    h_n = rir_generator(c_k, fs, mic_position, x_position, L, beta, n,
mtype, order, dim, orientation, hp_filter);
    mat_file1=sprintf('h_n.mat');
    save(mat_file1,'h_n')

    information = struct('beta', beta ,'x_position',  x_position,
'mic_position', mic_position, 'room_dim', L);

end
```

כדי הרשת תהיה מגוונת ותתאים למגוון רחב של חדרים הגרלנו את מימדי החדר, את מיקום הרמקול ואת זמן ההדהוד.

מימדי החדר בציר x ו-y מוגרלים בטווח $[3,6][m]$ ציר z, גובה החדר, קבוע ובאורך $3[m]$. בנוסף, זמן ההדהוד מוגרל בטווח $[0.15,0.65][s]$ בקפיצות של $0.005[s]$.

מיקום המיקרופון נקבע כך שהמרחק בין האדם למיקרופון יהיה בין $30[cm]$ ל-$2[m]$.

דוגמא לתמונת החדר שמייצרים:



לאחר יצירת המערכת, אנחנו מבצעות קונבולוציה בין ההקלטה למערכת ומוסיפות רעש אקראי בעוצמה של $20[dB]$:

```
%% CONV + NOISE(SNR=20dB)
d = conv(x,h);
v = rand(length(d),1); %noise
dVSv = sqrt(sum((d).^2)./(sum((v).^2)*10^2)); %SNR = 20dB
v = dVSv*v;
SNR_dv = snr(d,v);

y = v+d;
```

לאחר מכן, לאות המוצא מבצעים התמרת STFT עם הפרמטרים הבאים:

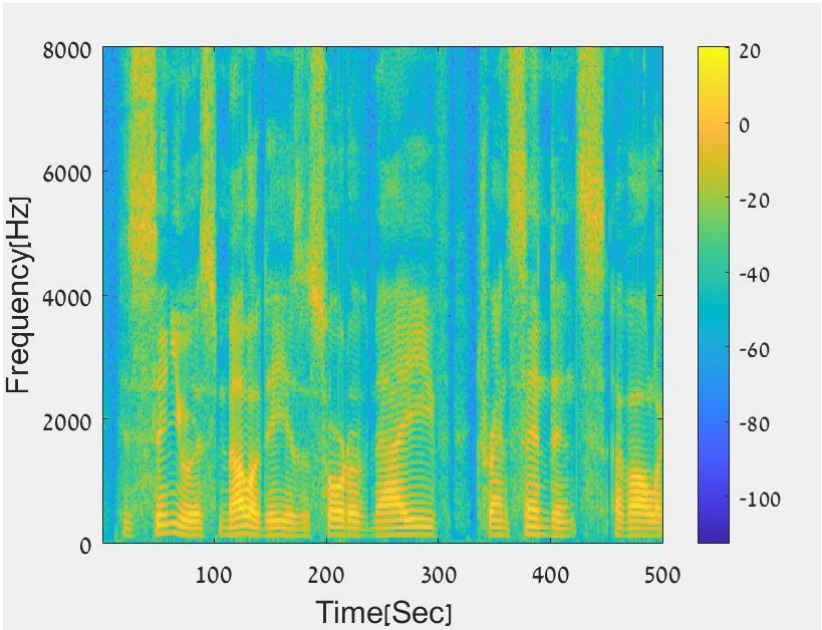
$$analysis\ window = hamming$$

$$window\ length = 512$$

$$overlap = 75\%$$

את ההתמרה חילקנו למקטעים של $4[s]$ באשר יצרנו חפיפה של $0.5[s]$ בין המקטעים השונים. בנוסף, לקחנו רק את התדרים החיובים $(0 - 8[kHz])$.

דוגמא לתמונת ה-STFT שמתקבלת:

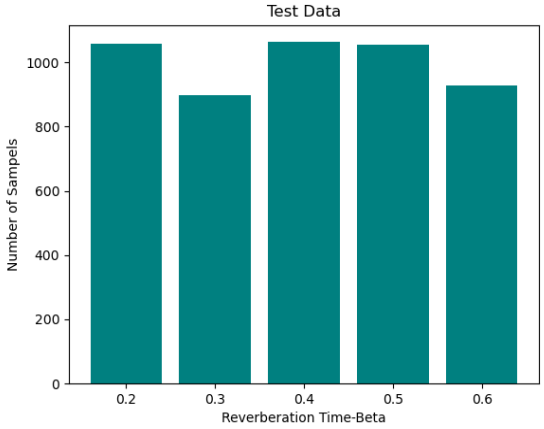


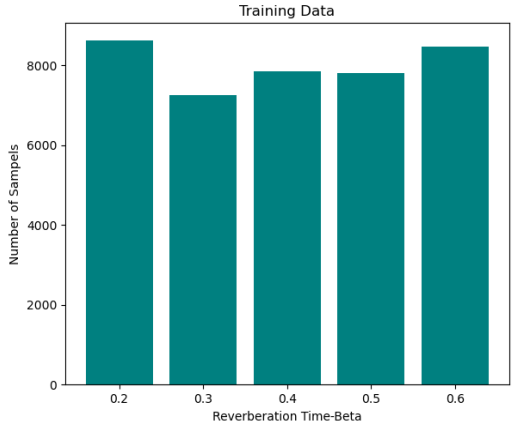
הגודל של מטריצת ה-STFT שמתקבלת: $(\text{freaquencies}, \text{frames}) = (257 \times 500)$

שמרנו כל תוצאה שמתקבלת בקובץ mat שמכיל את התמונה לפני (original) ואחרי הקונבולוציה (feature) ואת הפרמטרים של החדר:

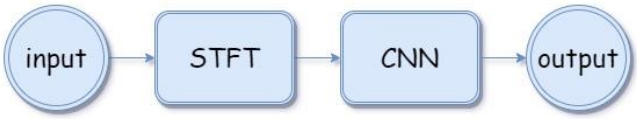
Import	Name	Size	Bytes	Class
☑	beta	1x1	8	double
☑	feature	257x500	1028000	double
☑	mic_position	1x3	24	double
☑	original	257x500	1028000	double
☑	room_dim	1x3	24	double
☑	x_position	1x3	24	double

בנוסף, וודאנו שכמות הדוגמאות שמכניסים לרשת מכל class פחות או יותר זהה גם עבור סט האימון וגם עבור סט הבדיקה:





בניית הרשת



פרמטרי הרשת

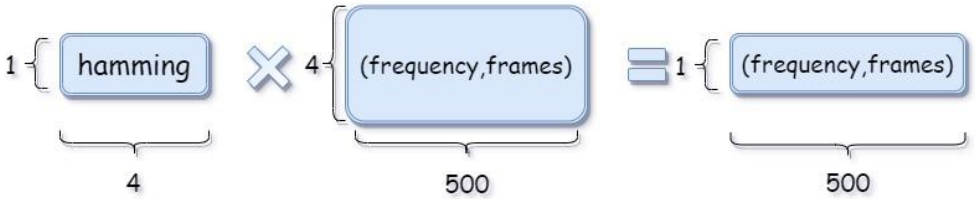
הרצנו את הרשת עם הפרמטרים הבאים:

```
#%% Parameters

epochs = 10
lr_rate = 0.0001
weight_decay = 0.001
batchSize = 64
```

הכניסה לרשת

הכניסה לרשת זה מטריצת ה-STFT שייצרנו כפי שהסברנו קודם אחרי שעברה עיבוד נוסף. כפי שציינו המטריצה בגודל $(\text{freaquencies}, \text{frames}) = (257 \times 500)$. כשמשתמשים ברשת CNN, המימדים של הקלט משפיעים על מספר הפרמטרים שמאמנים ובכך משפיעים על הסיבוכיות של המודל. כדי לשמור על מודל עם רמת סיבוכיות נמוכה תוך שמירה על המידע שאנחנו זקוקים לו כדי לשערך את ה-T_{60} בחרנו לבצע טרנספורמציה עם רזולוציה ספקטרלית נמוכה.

יצרנו חלון hamming מנורמל עם ארבעה מקדמים. הכפלנו את החלון בכל 4 תדרים סמוכים (בלי חפיפה) ומיצענו על התוצאה.



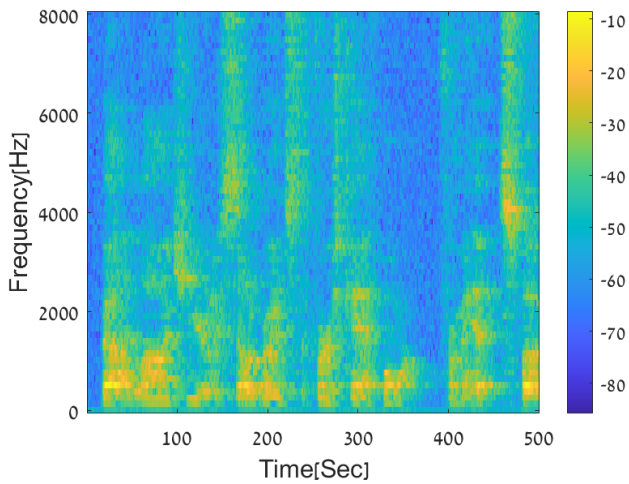
סה"כ הגודל של המטריצה הסופית שהיא ה-input לרשת:

$$(batch\ size, channels, freaquencies, frames) = (64\text{x}1\text{x}64\text{x}500)$$

הקוד שמימשנו:

```
#%% Function - Preprocesing to the STFT

def Squeeze_freq(stft,fs):
    squeeze_stft,matrix_stft = [],[]
    temp_stft = stft
    hamming4 = np.hamming(4)
    hamming4 = np.expand_dims(hamming4/sum(hamming4),axis = 0)
    for row in temp_stft:
        if (len(np.asarray(matrix_stft)) == 4):                    # Checks that we have reached the fourth row
            squeeze_stft.append(np.matmul(hamming4,matrix_stft))    # Multiply the matrix in hamming window
            matrix_stft=[row]
        else:
            matrix_stft.append(row)
    squeeze_stft = np.squeeze(squeeze_stft,axis = 1)
    return np.asarray(squeeze_stft)
```

דוגמא ל-input של הרשת:



שכבות הרשת

הרשת היא רשת CNN שמורכבת משלוש שכבות קונבולוציה עם פונקציית אקטיבציה ReLU, לאחר כל שכבה (לפני ה-ReLU) יש שכבת batch normalization. לאחר שכבות הקונבולוציה יש שכבה של drop-out עם הסתברות $\frac{1}{2}$, כדי למנוע overfitting.

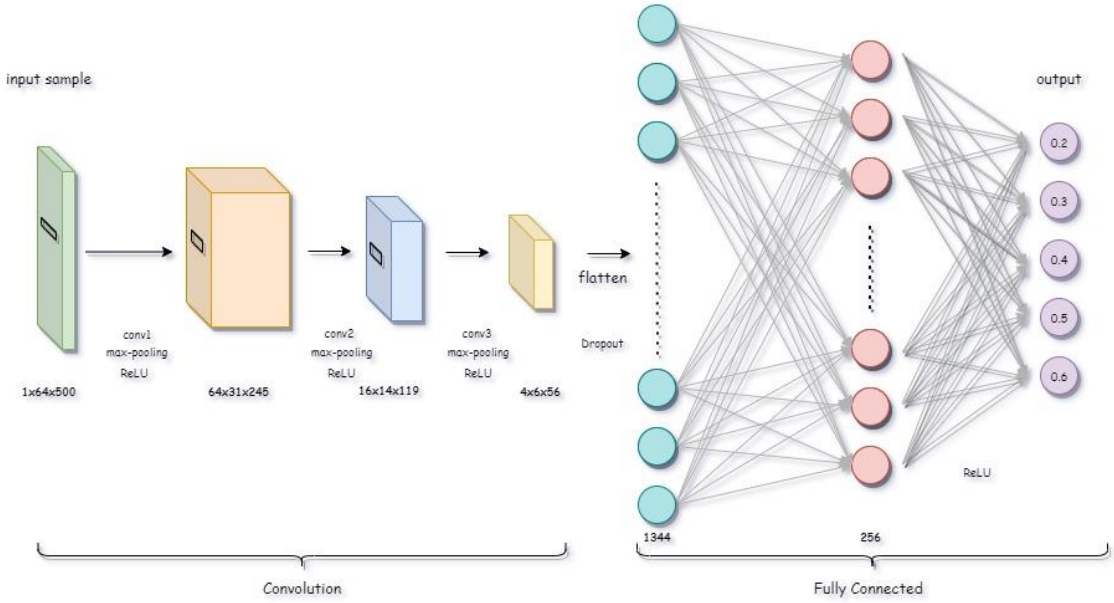
	Conv1	Conv2	Conv3
Size	3×10	3×8	3×8
Stride	(2,2)	(2,2)	(2,2)
Filters	64	16	4

לבסוף, יש שתי שכבות של fully connected עם פונקציית אקטיבציה ReLU ביניהן.

הקוד שמימשנו:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.cnn_layers = nn.Sequential(
            # Size = (batch size,channels,freaquencies,frames)
            # Defining a 2D convolution layer: Input Tensor: 64x1x64x500; Output Tensor: 64x64x31x245;
            nn.Conv2d(in_channels = 1, out_channels = 64, kernel_size = (3,10), stride = (1,1), padding = 0),
            nn.MaxPool2d((2, 2), stride=(2, 2)),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            # Defining another 2D convolution layer: Input Tensor: 64x64x31x245; Output Tensor: 64x16x14x119;
            nn.Conv2d(in_channels = 64, out_channels = 16, kernel_size = (3,8), stride = (1,1), padding = 0),
            nn.MaxPool2d((2, 2), stride=(2, 2)),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            # Defining another 2D convolution layer: Input Tensor: 64x16x14x119; Output Tensor: 64x4x6x56;
            nn.Conv2d(in_channels = 16, out_channels = 4, kernel_size = (3,8), stride = (1,1), padding = 0),
            nn.MaxPool2d((2, 2), stride=(2, 2)),
            nn.BatchNorm2d(4),
            nn.ReLU(),
            nn.Dropout(p = 0.5),
        )

        self.linear_layers = nn.Sequential(
            nn.Linear(in_features = 4*6*56, out_features = 256),  #Input: 64x1344; Output Tensor: 64x256;
            nn.ReLU(),
            nn.Linear(in_features = 256, out_features = 5),           #Input: 64x256; Output Tensor: 64x5;
            nn.Softmax()
        )
```

דיאגרמה של הארכיטקטורה שמימשנו עבור רשת הסיווג :



מוצא הרשת

במוצא הרשת נקבל הערכה ל-T_{60} של החדר שממנו התקבלה ההקלטה. לכל כניסה של הרשת, שזה למעשה קטע קול באורך של $4[s]$, הרשת מייצרת הערכה ל-T_{60}.

הערה: כפי שציינו בהקדמה, במהלך הפרויקט בנינו שתי רשתות קונבולוציה שהמוצא שלהם שונה. רשת אחת מבצעת סיווג והרשת השנייה מבצעת רגרסיה. נפרט תחילה על הרשת שמבצעת סיווג ונראה את התוצאות שקיבלנו לאח"כ נפרט על הרשת שמבצעת רגרסיה.

בהתחלה בנינו רשת שמבצעת סיווג. כלומר, המוצא של הרשת היה ה-class שמייצג את זמן ההדהוד שהכי קרוב לזמן האמיתי. עבדנו עם חמישה מחלקות עם זמני הדהוד: $0.2,0.3,0.4,0.5,0.6[s]$. כדי לבצע את הסיווג הוספנו למוצא הרשת שכבה נוספת - *softmax* , פונקציה זו מתאימה לבעיות סיווג.

תוצאות

תוצאות עבור רשת הסיווג

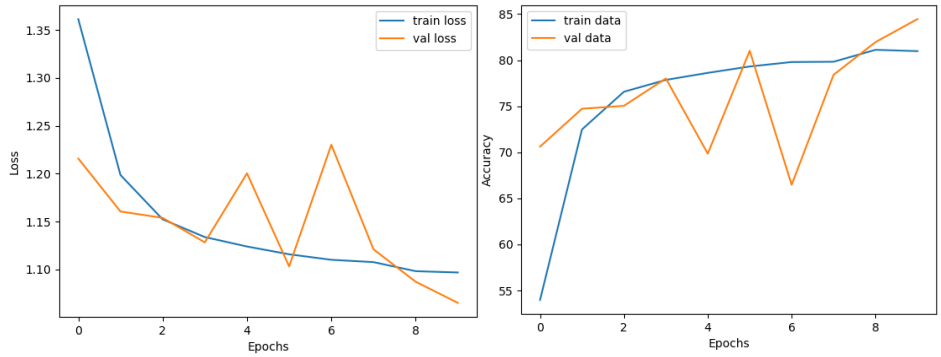
שלב אימון הרשת

בדי לאמן את הרשת עבדנו עם פונקציית מחיר: cross-entropy.

הקוד שמימשנו:

```
#%% Training
train_loss, val_loss = [], []
train_acc, val_acc = [], []
min_val_loss,max_acc = 1000,0

for epoch in range(int(epochs)):

    epoch_train_loss, epoch_val_loss = 0 ,0
    epoch_train_acc, epoch_val_acc = 0 ,0
    model.train()

    for i, (images, labels) in enumerate(train_loader): # on batch
        images = images.to(device)
        labels = labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()
        # forward & backward & optimize
        outputs = model(images)
        loss = criterion(outputs, labels) # single loss for each batch
        loss.backward()
        optimizer.step()

        # accuracy & loss
        epoch_train_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1) #sofftmax
        acc = (predicted == labels).sum()
        epoch_train_acc += acc
    # over all there is N files for train; we have 15 epochs; in each epoch N/64 batches; in each batch 64 files
    train_loss.append(epoch_train_loss/len(train_loader))        # divide by number of batches N/64, to get mean loss for each epoch
    train_acc.append(100*(epoch_train_acc.item()/ len(train_x))) # divide by number of files N, to get acc for each epoch


    # VAL
    model.eval()

    with torch.no_grad():
        for i, (val_images, val_labels) in enumerate(val_loader):
            val_images = val_images.to(device)
            val_labels = val_labels.to(device)
            # forward
            val_outputs = model(val_images)
            # accuracy & loss
            loss = criterion(val_outputs, val_labels)
            _,predicted = torch.max(val_outputs.data, 1) #sofftmax
            epoch_val_acc += ((predicted == val_labels).sum())
            epoch_val_loss += loss.item()
    # over all there is N files for train; we have 15 epochs; in each epoch N/64 batches; in each batch 64 files
    val_loss.append(epoch_val_loss / len(val_loader))      # divide by number of batches N/64, to get mean loss for each epoch
    val_acc.append(100*(epoch_val_acc.item()/ len(val_x))) # divide by number of files N, to get acc for each epoch
    if (val_loss[epoch]<min_val_loss):
                model_copy = copy.deepcopy(model)
                min_val_loss=val_loss[epoch]
                max_acc=val_acc[epoch]
    print(f"Epoch {epoch+1}/{epochs}, train_loss: {train_loss[epoch]},train_acc: {train_acc[epoch]}, val_loss: {val_loss[epoch]},val_acc: {val_acc[epoch]}")
```

התוצאות שהתקבלו:



ניתן לראות שבכל שמספר ה-epoch-ים עולה כך הדיוק וה-loss של סט האימון משתפר. בנוסף, ניתן לראות בעזרת סט הוולידציה שהרשת לא מגיעה ל-overfitting והיא עוקבת בצורה טובה אחרי סט האימון.

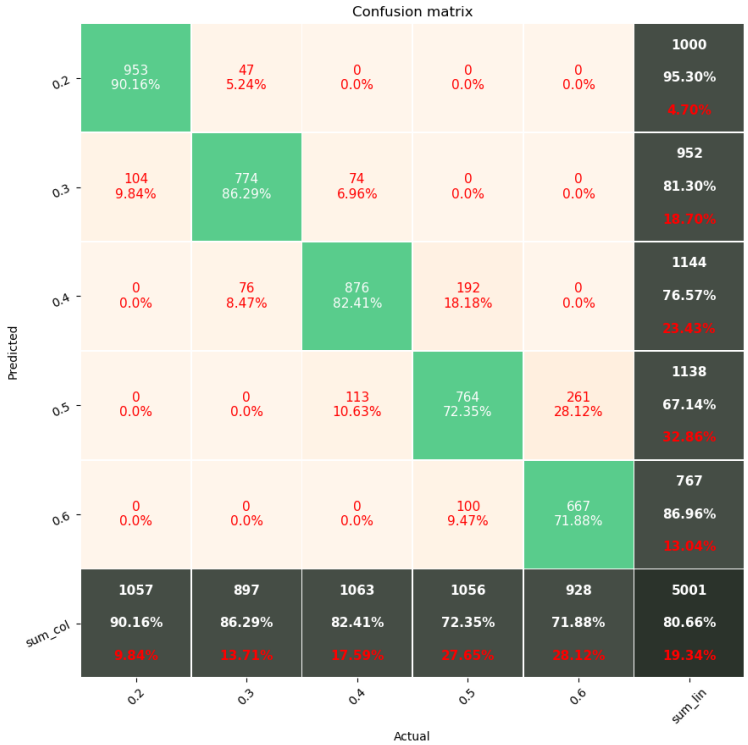
שלב בדיקת ביצועי הרשת

את הביצועים של הרשת בדקנו על סט הבדיקה עם המשקולות שקיבלנו עבורם את הדיוק הגבוה ביותר בסט הוולידציה.

הקוד שמימשנו:

```
#%%Test

y_pred,y_true=[],[]
test_acc = 0
model.eval()

with torch.no_grad():
    for batch_idx, (data, target) in enumerate(test_loader):
        data, target = data.to(device), target.to(device)
        # forward
        outputs = model(data)
        # accuracy
        predicted = torch.max(outputs.data, 1)[1]
        test_acc += (predicted == target).sum()
        y_true = y_true+target.tolist()
        y_pred = y_pred+predicted.tolist()

print('Accuracy of the network on the test images: %d %%' % (100 * test_acc //len(test_data)))
```

התוצאות שהתקבלו:



ניתן לראות שהרשת מגיעה לדיוק של בערך 81%. בנוסף, ניתן לראות שגם כשהרשת מבצעת טעות בסיווג, השגיאה שמתקבלת היא $\pm 0.1[s]$.

בנוסף, בדקנו את ביצועי הרשת על עוד סט בדיקה. את סט הבדיקה ייצרנו כמו שהסברנו קודם עם ההבדל הבא: במקום להגריל מערכת ע"י שימוש ב-rir generator, השתמשנו בתגובות תדר מהאתר של פרופסור שרון גנות (Index of /~gannot/RIR_DATABASE (biu.ac.il)). תגובות התדר שעשינו בהם שימוש:

- Impulse_response_Acoustic_Lab_Bar-Ilan_University_(Reverberation_0.160s)_3-3-3-8-3-3-3_1m_000
- Impulse_response_Acoustic_Lab_Bar-Ilan_University_(Reverberation_0.360s)_3-3-3-8-3-3-3_1m_285
- Impulse_response_Acoustic_Lab_Bar-Ilan_University_(Reverberation_0.610s)_3-3-3-8-3-3-3_1m_000
- Impulse_response_Acoustic_Lab_Bar-Ilan_University_(Reverberation_0.160s)_3-3-3-8-3-3-3_2m_345
- Impulse_response_Acoustic_Lab_Bar-Ilan_University_(Reverberation_0.360s)_3-3-3-8-3-3-3_2m_345
- Impulse_response_Acoustic_Lab_Bar-Ilan_University_(Reverberation_0.610s)_3-3-3-8-3-3-3_2m_345

התוצאות שהתקבלו:

ניתן לראות, שהרשת לא תמיד מבצעת סיווג נכון עבור ה-T_{60} אבל היא כן שומרת על אחידות בתוצאה עבור כל הדוגמאות עם אותו זמן הדהוד. בנוסף, ניתן לראות שהטעות היא רק במחלקה אחת, נמוכה יותר. כלומר, טעות של $-0.1[s]$.

ניתן לראות שהצלחנו להגיע ברשת לביצועים טובים. אולם, כאשר הרשת מבצעת סיווג היא יכולה להעריך רק חמישה זמנים שונים, למרות שבמציאות זמני ההדהוד יכולים להיות הרבה יותר מגוונים. לכן, בנינו רשת נוספת שמבצעת רגרסיה. כלומר, הרשת מחזירה מספר רציונאלי שמייצג את זמן ההדהוד של החדר.

תוצאות עבור רשת הרגרסיה

עבור הרגרסיה השתמשנו ב-database זהה ושכבות הרשת היו זהות. ההבדלים הם בפונקציית העלות (loss function) שהיא MAE במקום cross entropy (שינוי זה משנה מעט את הקוד במהלך אימון/בדיקה) ובמוצא של הרשת.

את פונקציית העלות (loss function) הגדרנו באופן הבא:

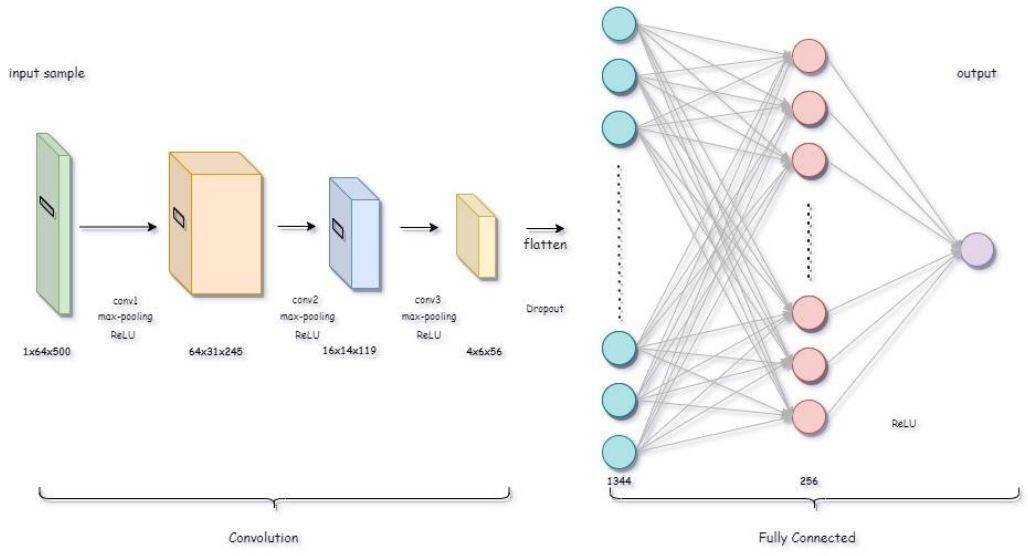
$$\ell(prediction, label) = \frac{abs(prediction - label)}{0.1}$$

בגלל שהבדל של $0.1[s]$ ברשת הסיווג היה מסווג למחלקה לא נכונה חילקנו את ה-loss ב-0.1 כדי להבטיח שרשת הרגרסיה תגיע לשגיאות נמוכות יותר מ-$0.1[s]$.

הרשת:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.cnn_layers = nn.Sequential(
            # Size = (batch size,channels,freaquencies,frames)
            # Defining a 2D convolution layer: Input Tensor: 64x1x64x500; Output Tensor: 64x64x31x245;
            nn.Conv2d(in_channels = 1, out_channels = 64, kernel_size = (3,10), stride = (1,1), padding = 0),
            nn.MaxPool2d((2, 2), stride=(2, 2)),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            # Defining another 2D convolution layer: Input Tensor: 64x64x31x245; Output Tensor: 64x16x14x119;
            nn.Conv2d(in_channels = 64, out_channels = 16, kernel_size = (3,8), stride = (1,1), padding = 0),
            nn.MaxPool2d((2, 2), stride=(2, 2)),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            # Defining another 2D convolution layer: Input Tensor: 64x16x14x119; Output Tensor: 64x4x6x56;
            nn.Conv2d(in_channels = 16, out_channels = 4, kernel_size = (3,8), stride = (1,1), padding = 0),
            nn.MaxPool2d((2, 2), stride=(2, 2)),
            nn.BatchNorm2d(4),
            nn.ReLU(),
            nn.Dropout(p = 0.5),
        )

        self.linear_layers = nn.Sequential(
            nn.Linear(in_features = 4*6*56, out_features = 256),      #Input: 64x1344; Output Tensor: 64x256;
            nn.ReLU(),
            nn.Linear(in_features = 256, out_features = 1),           #Input: 64x256; Output Tensor: 64x1;
        )
```

דיאגרמה של הארכיטקטורה שמימשנו עבור רשת הרגרסיה:

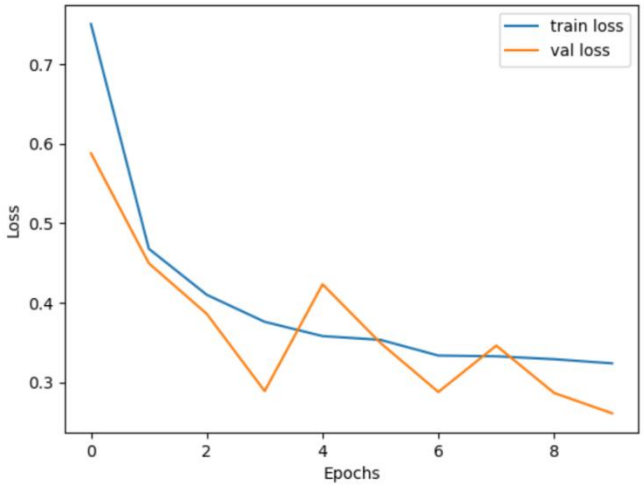


שלב אימון הרשת

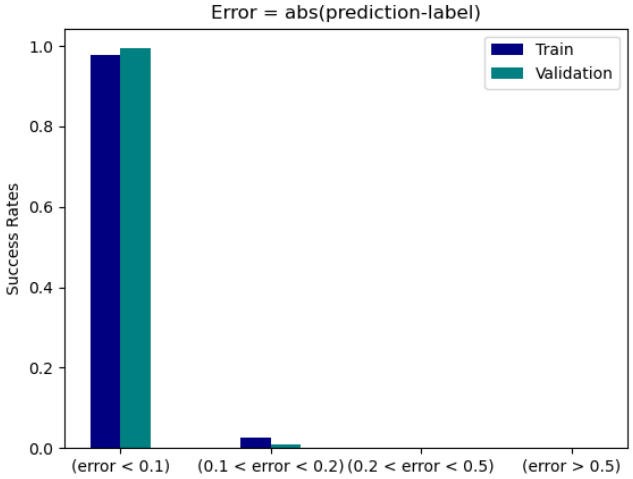
הקוד שמימשנו:

```
#%% Training
train_loss, val_loss = [],[]
train_successful, train_low_error,train_med_error, train_high_error = [],[],[],[]
val_successful, val_low_error,val_med_error, val_high_error = [],[],[],[]
MSE, MSE_val = 0,0
min_val_loss = 1000

for epoch in range(int(epochs)):

    epoch_train_loss, epoch_val_loss= 0 ,0
    successful, low_error,med_error, high_error = 0,0,0,0
    model.train()

    for i, (images, labels) in enumerate(train_loader): # on batch
        images = images.to(device)
        labels = labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()
        # forward & backward & optimize
        outputs = model(images)
        loss = criterion(outputs.float(), labels.float()) # single loss for each batch
        loss.backward()
        optimizer.step()


        # accuracy & loss
        MSE = abs(outputs.T-labels)
        epoch_train_loss += loss.item()
        successful += sum(sum(MSE < 0.1))
        low_error += (sum(sum(MSE < 0.2)) - sum(sum(MSE < 0.1)))
        med_error += (sum(sum(MSE < 0.5)) - sum(sum(MSE < 0.2)))
        high_error += (sum(sum(MSE >= 0.5)))

    # over all there is N files for train; we have 15 epochs; in each epoch N/64 batches; in each batch 64 files
    train_loss.append(epoch_train_loss/len(train_loader))       # divide by number of batches N/64, to get mean loss for each epoch
    train_successful.append(successful/len(train_x))            # divide by number of files N, to get acc for each epoch
    train_high_error.append(high_error/len(train_x))            # divide by number of files N, to get acc for each epoch
    train_med_error.append(med_error/len(train_x))              # divide by number of files N, to get acc for each epoch
    train_low_error.append(low_error/len(train_x))              # divide by number of files N, to get acc for each epoch

    # VAL
    model.eval()

    successful, low_error,med_error, high_error = 0,0,0,0

    with torch.no_grad():
        for i, (val_images, val_labels) in enumerate(val_loader):
            val_images = val_images.to(device)
            val_labels = val_labels.to(device)
            # forward
            val_outputs = model(val_images)
            # accuracy & loss
            loss = criterion(val_outputs, val_labels)
            epoch_val_loss += loss.item()
            MSE_val = abs(val_outputs.T-val_labels)
            successful += sum(sum(MSE_val < 0.1))
            low_error += (sum(sum(MSE_val < 0.2)) - sum(sum(MSE_val < 0.1)))
            med_error += (sum(sum(MSE_val < 0.5)) - sum(sum(MSE_val < 0.2)))
            high_error += (sum(sum(MSE_val >= 0.5)))

    # over all there is N files for train; we have 15 epochs; in each epoch N/64 batches; in each batch 64 files
    val_loss.append(epoch_val_loss/len(val_loader))             # divide by number of batches N/64, to get mean loss for each epoch
    val_successful.append(successful/len(val_x))                # divide by number of files N, to get acc for each epoch
    val_high_error.append(high_error/len(val_x))                # divide by number of files N, to get acc for each epoch
    val_med_error.append(med_error/len(val_x))                  # divide by number of files N, to get acc for each epoch
    val_low_error.append(low_error/len(val_x))                  # divide by number of files N, to get acc for each epoch

    if (val_loss[epoch]<min_val_loss):
        model_copy = copy.deepcopy(model)
        min_val_loss=val_loss[epoch]

    print(f"Epoch {epoch+1}/{epochs}, train_loss: {train_loss[epoch]}, val_loss: {val_loss[epoch]}")
```

התוצאות שהתקבלו:



ניתן לראות שבכל שמספר ה-epoch-ים עולה כך ה-loss של סט האימון משתפר. בנוסף, ניתן לראות בעזרת סט הוולידציה שהרשת לא מגיעה ל-overfitting והיא עוקבת בצורה טובה אחרי סט האימון.



ניתן לראות, שכמעט בכל הדוגמאות הרשת מצליחה להגיע לדיוק של פחות מ-$0.1[s]$ בשערוך זמן ההדהוד של החדר (T_{60}) גם בסט האימון וגם בסט הוולידציה.

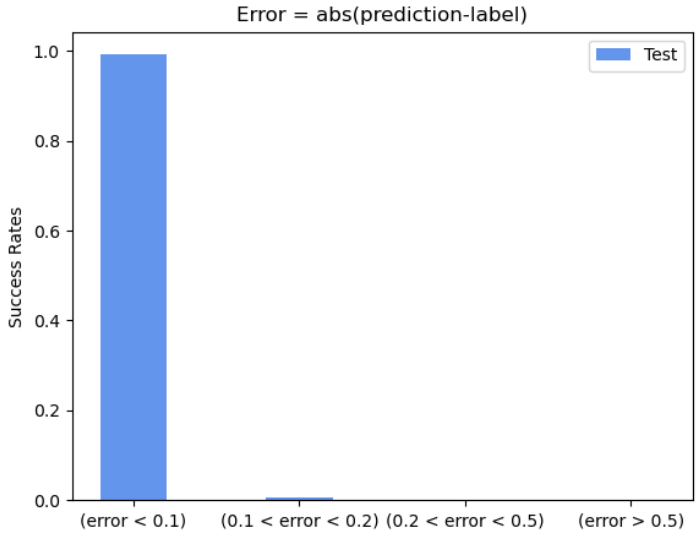
שלב בדיקת ביצועי הרשת

את הביצועים של הרשת בדקנו על סט הבדיקה עם המשקולות שקיבלנו עבורם את ה-loss הנמוך ביותר בסט הוולידציה.

הקוד שמימשנו:

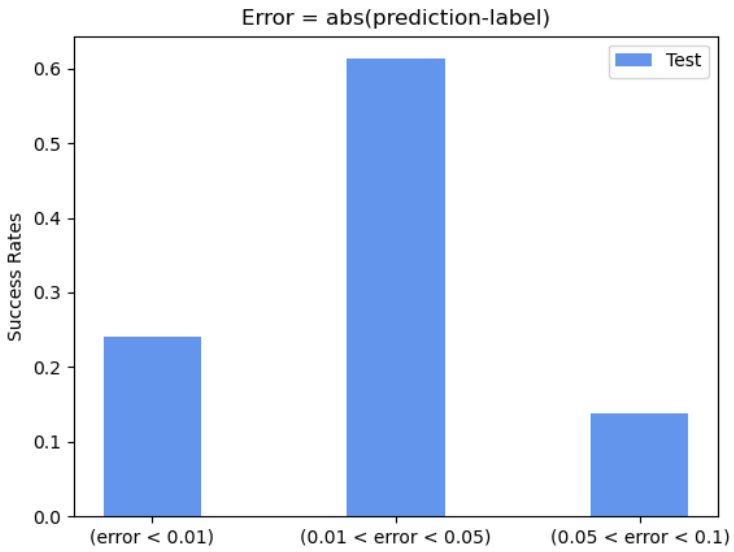
```
#%%Test

test_loss=[]
test_successful, test_low_error,test_med_error, test_high_error = [],[],[],[]
epoch_test_loss = 0
successful, low_error,med_error, high_error = 0,0,0,0
MSE_test = 0
y_pred,y_true=[],[]

model.eval()

with torch.no_grad():
    for i, (data, target) in enumerate(test_loader):
        data = data.to(device)
        target = target.to(device)
        # forward
        test_outputs = model(data)
        # accuracy & loss
        loss = criterion(test_outputs, target)
        epoch_test_loss += loss.item()
        MSE_test = abs(test_outputs.T-target)
        successful += sum(sum(MSE_test < 0.1))
        low_error += (sum(sum(MSE_test < 0.2)) - sum(sum(MSE_test < 0.1)))
        med_error += (sum(sum(MSE_test < 0.5)) - sum(sum(MSE_test < 0.2)))
        high_error += (sum(sum(MSE_test >= 0.5)))

        y_true = y_true+target.tolist()
        y_pred = y_pred+test_outputs.tolist()

# over all there is N files for train; we have 15 epochs; in each epoch N/64 batches; in each batch 64 files
test_loss.append(epoch_test_loss/len(test_loader))            # divide by number of batches N/64, to get mean loss for each epoch
test_successful.append(successful/len(test_data))             # divide by number of files N, to get acc for each epoch
test_high_error.append(high_error/len(test_data))             # divide by number of files N, to get acc for each epoch
test_med_error.append(med_error/len(test_data))               # divide by number of files N, to get acc for each epoch
test_low_error.append(low_error/len(test_data))               # divide by number of files N, to get acc for each epoch
```

התוצאות שהתקבלו:



ניתן לראות, שגם בסט הבדיקה כמעט בכל הדוגמאות הרשת מצליחה להגיע לדיוק של פחות מ-$0.1[s]$ בשערוך זמן ההדהוד של החדר (T_{60}).

כדי לקבל הבנה יותר טובה של התוצאות שקיבלנו חילקנו את הדוגמאות שהתקבלה עבורם שגיאה שנמוכה מ- $0.1[s]$ לרזולוציה קטנה יותר:



עבור כמעט 85% מהדוגמאות הרשת מצליחה לשערך את ה-T_{60} של החדר בדיוק של פחות מ-$0.05[s]$.

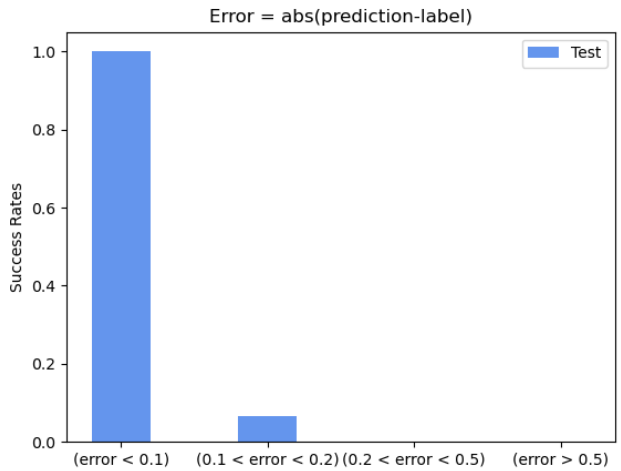
כדי להשוות את הביצועים שקיבלנו ברשת הזו לביצועים שקיבלנו ברשת הקודמת עיגלנו את השערוך שהתקבל בזמני ההדהוד השונים על מנת לסווג אותם לחמש קטגוריות ויצרנו confusion matrix כמו קודם.

התוצאות שהתקבלו:

ניתן לראות שהרשת מגיעה לדיוק של בערך 76% כאשר הרשת הקודמת הגיעה ל-81%. רשת הסיווג התאמנה לעשות סיווג בין המחלקות השונות לעומת זאת, הרגרסיה לא התאמנה על הקוונטיזציה שביצענו. יכול להיות שבדיוק באזור החפיפה רשת הרגרסיה נופלת כי לא התאמנו על כך. ולכן למרות שמספר האחוזים נמוך יותר רשת הרגרסיה טובה יותר מרשת הסיווג. בנוסף, כפי שראינו בתמונת ההיסטוגרמה האחרונה הרשת מצליחה להגיע ברוב התמונות לדיוק של ± 0.05. שגיאה בגודל הזה עבור ה-T_{60} במערכות שמימשנו בחלק הראשון כמעט לא משפיעה על התוצאות. עוד יתרון של הרשת הזו כפי שציינו קודם שניתן לקבל כל ערך של T_{60} בין $0.2[s]$ ל-$0.6[s]$ לא רק מספרים שלמים.

כמו שעשינו עבור הרשת הקודמת, בדקנו את ביצועי הרשת הזו על עוד סט בדיקה שייצרנו כפי שהסברנו קודם.

התוצאות שהתקבלו:



ניתן לראות שוב שכמעט בכל הדוגמאות הרשת מצליחה להגיע לדיוק של פחות מ-$0.1[s]$ בשערוך זמן ההדהוד של החדר (T_{60}).

ניתן לראות, שהתוצאות די דומות לביצועים שהתקבלו עבור הרשת הקודמת. גם פה הרשת לא תמיד מבצעת סיווג נכון עבור ה-T_{60} אבל היא כן שומרת על אחידות בתוצאה עבור כל הדוגמאות עם אותו זמן הדהוד. בנוסף, ניתן לראות שהטעות היא רק במחלקה אחת, נמוכה יותר. כלומר, טעות של $-0.1[s]$.

הערה: הקוד המלא של שני הרשתות מצורף בקובץ zip לפרויקט.

סיכום חלק ב

בחלק זה בנינו שני מודלים שונים: רשת שמבצעת סיווג ורשת שמבצעת רגרסיה. שתי הרשתות מגיעות לביצועים טובים. עם זאת לרשת הרגרסיה יש יתרון על פני רשת הסיווג כפי שציינו קודם.

חלק ג: שילוב המערכות

המערכת הכוללת:

כדי לבדוק את ביצועי המערכת הכוללת שרשרנו קטעי קול של דובר אחד ממאגר המידע של TIMIT-CD לאורך כולל של כדקה. כאשר הגרלנו דובר מסוים בצורה אקראית ושרשרנו מספר קטעי דיבור שלו. את קטע הקול סיננו עם תגובת התדר של החדר הבא (מהאתר של פרופסור שרון גנות):

Impulse_response_Acoustic_Lab_Bar-Ilan_University_(Reverberation_0.360s)_3-3-3-8-3-3-3_1m_000

כאשר זמן ההדהוד במקרה זה שווה ל-$0.36[s]$.

כפי שהסברנו בחלק הקודם, לקחנו את קטע הקול וביצענו לו התמרת STFT. הוצאנו תמונת זמן תדר לכל מקטע באורך של $4[s]$.

את התמונות הכנסנו לרשת הנוירונים שאומנה בחלק הקודם כאשר שמרנו את המשקולות האופטימליים. קיבלנו וקטור ערכים עבור זמן ההדהוד של החדר לכל תמונה.

עדכנו את אלגוריתם הסינון כך שיהיה מסתגל לשינוי בהערכת זמן ההדהוד. למעשה, אנחנו מדמות מערכת שפועלת בזמן אמת. תחילה, האלגוריתם מאותחל עם זמן הדהוד נמוך של $0.1[s]$ - במקרה כזה אין היפוך של מטריצות גדולות. לאחר מכן האלגוריתם מתעדכן בהתאם לזמן ההדהוד המשוערך שמתקבל מרשת הנוירונים.
לחילופין, ניתן לאתחל את זמן ההדהוד בערך גבוה ואז להוריד אותו בהתאם למוצא הרשת.
האלגוריתם מקבל את זמן ההדהוד כווקטור של ערכים ומעדכן את זמן ההדהוד של האלגוריתם בכל 20 שניות.
העדכון מתבצע ע"י לקיחת הערך הממוצע של זמני ההדהוד שהתקבלו ב-20 השניות האחרונות.

כדי לבדוק את ביצועי המערכת הכוללת עשינו בדיקה דומה לבדיקה שעשינו בחלק א'.

תחילה, בדקנו את מוצא ה-AEC כאשר עדכנו את מספר הטפים בהתאם ל- T_{60} . כדי לקבוע את מספר הטפים שמתאים ל-T_{60} שמתקבל מהרשת, הסתמכנו על התוצאות שהתקבלו בסיכום החלק הראשון:

$\boldsymbol{T_{60}}$	$\mathbf{0.2}[s]$	$\mathbf{0.3}[s]$	$\mathbf{0.4}[s]$	$\mathbf{0.5}[s]$	$\mathbf{0.6}[s]$
$Number\ of\ Tapes$	8	16	16	32	**32**

כאשר לקחנו עבור כל זמן הדהוד את מספר הטפים שהתקבל עבורו SNR מקסימלי.
לאחר מכן, בדקנו את מוצא ה-PF כאשר מספר הטפים היה קבוע על 4 ועדכנו רק את ה-T_{60} של המסנן.

תוצאות

בדיקת תקינות המערכת המשולבת - ערכים שרירותיים של T_{60}

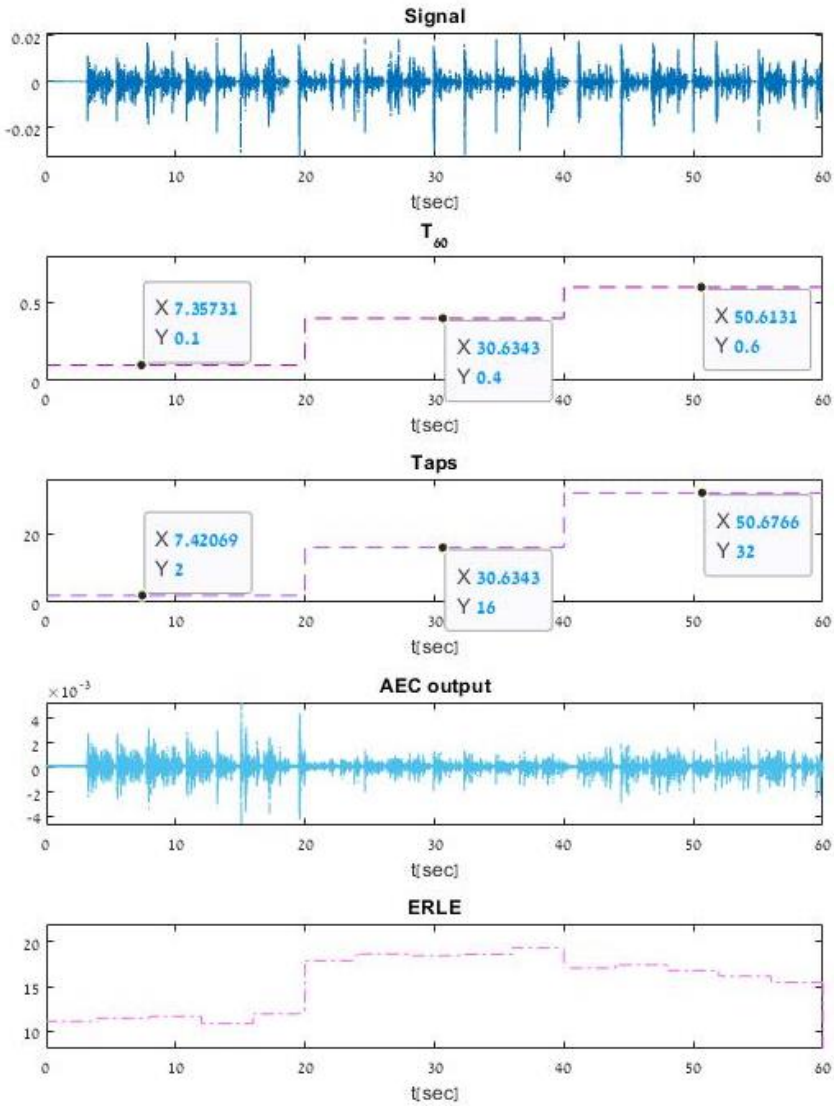
תחילה, בדקנו את ביצועי המערכת המסתגלת לשינוי בהזנת הערך T_{60} עבור הזנת וקטור אקראי למערכת.
כאשר כל איבר בווקטור מציין זמן הדהוד עבור תמונת זמן-תדר באורך $4[s]$.
בחרנו את הווקטור הבא:

0.4	0.4	0.4	0.4	0.4	0.4	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6

כאשר זמן ההדהוד האמיתי של החדר הינו $0.36[s]$.

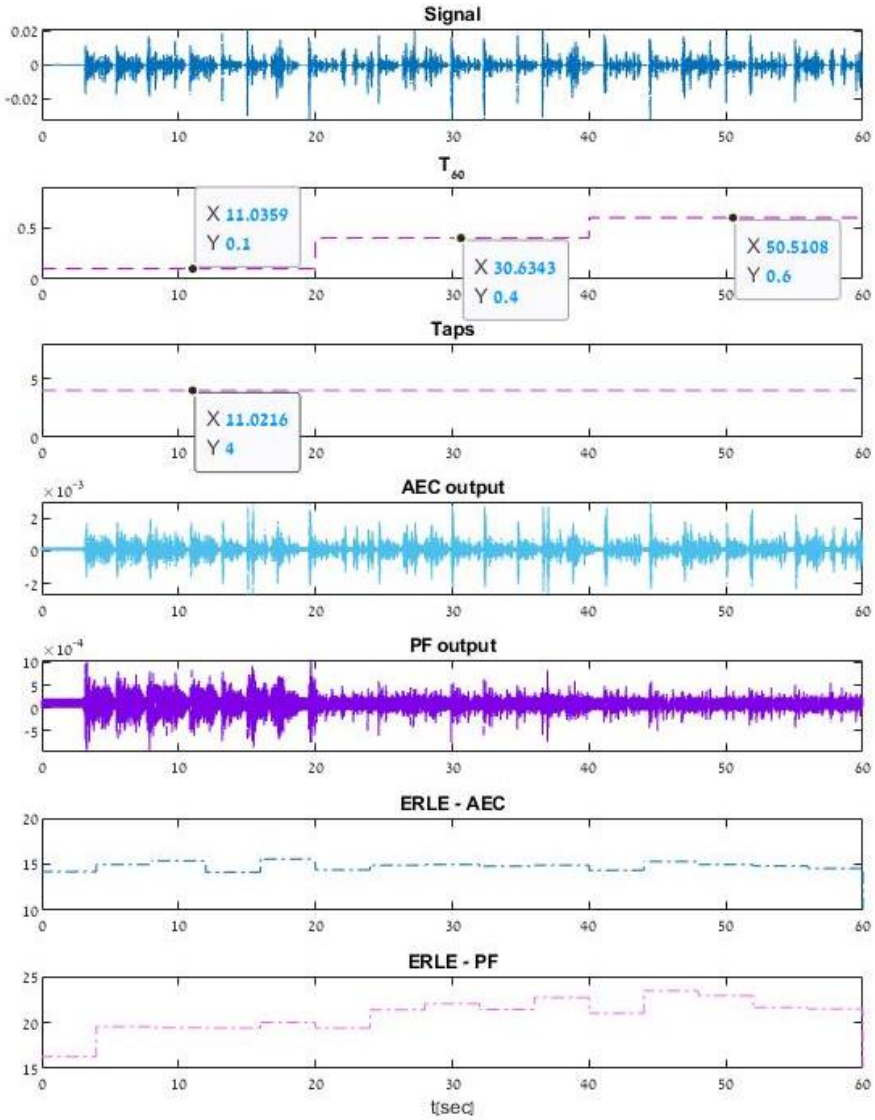
נזכיר כי בהתחלה המערכת מאותחלת על הערך $T_{60} = 0.1[s]$ עבור 20 השניות הראשונות.

מוצא הAEC:



ניתן לראות שקיבלנו את הביצועים הטובים ביותר כאשר זמן ההדהוד שהכנסנו למערכת שווה ל- $0.4[s]$. שזה אכן הערך הקרוב ביותר לערך האמיתי. תחילה אתחלנו את המערכת בזמן הדהוד נמוך יותר-$0.1[s]$ וההנחתה הייתה נמוכה, לאחר מכן כשהגדלנו את זמן ההדהוד לערך המתאים קיבלנו את התוצאות הטובות ביותר וכאשר הגדלנו את זמן ההדהוד מעבר לזמן האמיתי קיבלנו שהביצועים התקלקלו.

מוצא הPost-Filter:



ניתן לראות שההנחתה עבור מוצא ה-AEC יחסית קבועה וסביב הערך $15[dB]$. הסיבה שה-*ERLE* לא קבוע נובע כנראה מרצף הדיבור ומרגעים בהם יש רעש בלבד.

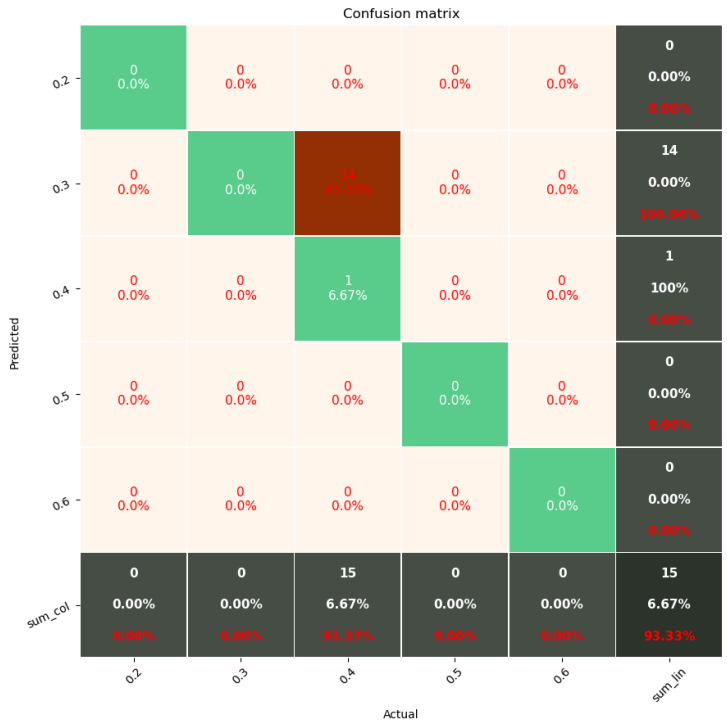
במוצא ה-*PF* ניתן לראות שינויים בהנחתה עבור זמני הדהוד שונים. כאשר ערך ה-T_{60} שהכנסנו למערכת היה נמוך מהערך האמיתי קיבלנו שההנחתה נמוכה. כאשר הגדלנו את ה-T_{60} לערך הקרוב לערך האמיתי קיבלנו שיפור ניכר בהנחתה. כאשר הגדלנו את ה-T_{60} מעבר לערך האמיתי ניתן לראות שההנחתה לא נפגעה. דבר זה הגיוני, כפי שראינו בסיכום עבור החלק הראשון כאשר אנחנו קובעות שמספר הטפים קבוע ושווה ל-4 ככל שנגדיל את ה-T_{60} מעבר לזמן האמיתי נקבל שהביצועים לא נפגעים. זה נובע כנראה מהעובדה שנשאר הדהוד רב לאחר ה-*AEC*. ומספר הטפים נמוך ביחס לזמן ההדהוד האמיתי של החדר.

בדיקת ביצועי המערכת המשולבת-ערכים מרשת הסיווג של T_{60}

בשלב זה בדקנו את ביצועי המערכת כאשר הערך של ה-T_{60} שהכנסנו לאלגוריתם הוא מהווקטור המשוערך שקיבלנו מרשת הסיווג.

הכנסנו את תמונות ה-*STFT* של קטע הקול שבנינו לרשת הסיווג המאומנת מחלק ב'.

התוצאות שהתקבלו:



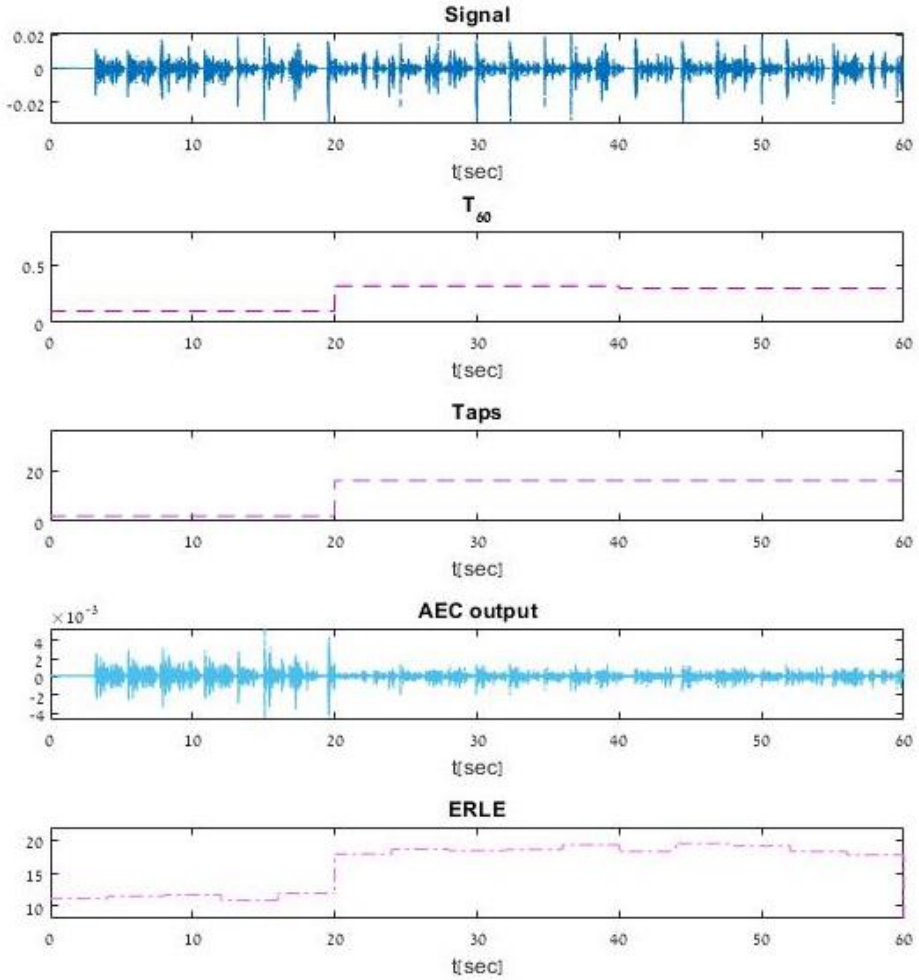
ניתן לראות שהתוצאות שהתקבלו תואמות לתוצאות שקיבלנו בחלק הקודם. יש אחידות בשערוך, פרט לתמונה אחת.

נציין כי במקרה שלנו זמן ההדהוד האמיתי הינו $T_{60} = 0.36[s]$. לכן, במקרה זה הטעות בשערוך נמוכה מ$0.1[s]$.

קיבלנו את הווקטור הבא:

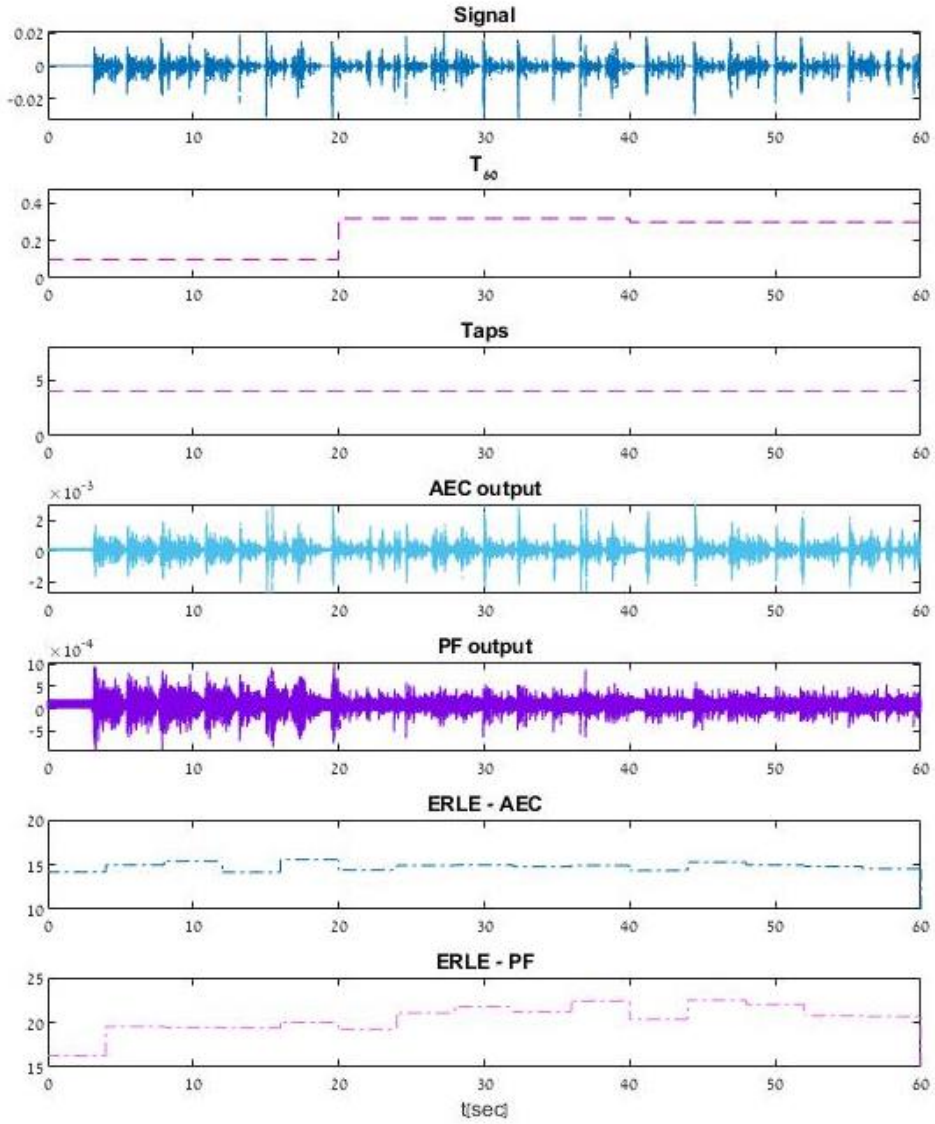
0.3	0.3	0.3	0.3	0.4	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3

הכנסנו את הווקטור לאלגוריתמים שמימשנו וקיבלנו את התוצאות הבאות:
מוצא הAEC:



ניתן לראות שיש שיפור ניכר בביצועים כאשר זמן ההדהוד שאנחנו מכניסים מתאים לזמן ההדהוד במוצא הרשת. נזכור כי אנחנו ממצעים על מספר ערכים לכן, לאחר 20 שניות ראשונות ה-T_{60} היה קצת יותר גבוה. במקרה זה, עבור שני זמני ההדהוד שקיבלנו מספר הטפים היה זהה לכן עבור ה-AEC לא נראה שינוי ניכר בביצועים.

מוצא הPF:



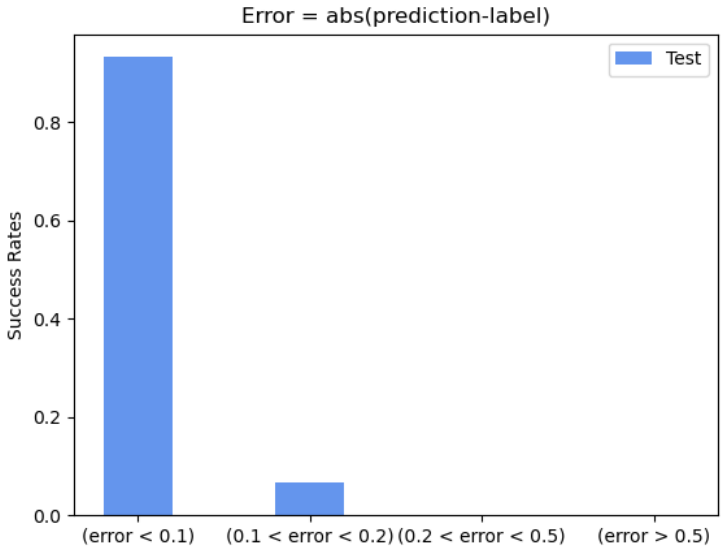
ניתן לראות שבצורה דומה למה שקיבלנו קודם, ההנחתה עבור מוצא ה-AEC יחסית קבועה וסביב הערך $15[dB]$.
עבור ה-*PF* ניתן לראות שכאשר זמן ההדהוד נמוך קיבלנו הנחתה נמוכה יותר וכאשר הגדלנו את זמן ההדהוד קיבלנו שההנחתה השתפרה.

בדיקת ביצועי המערכת המשולבת - ערכים מרשת הרגרסיה של T_{60}

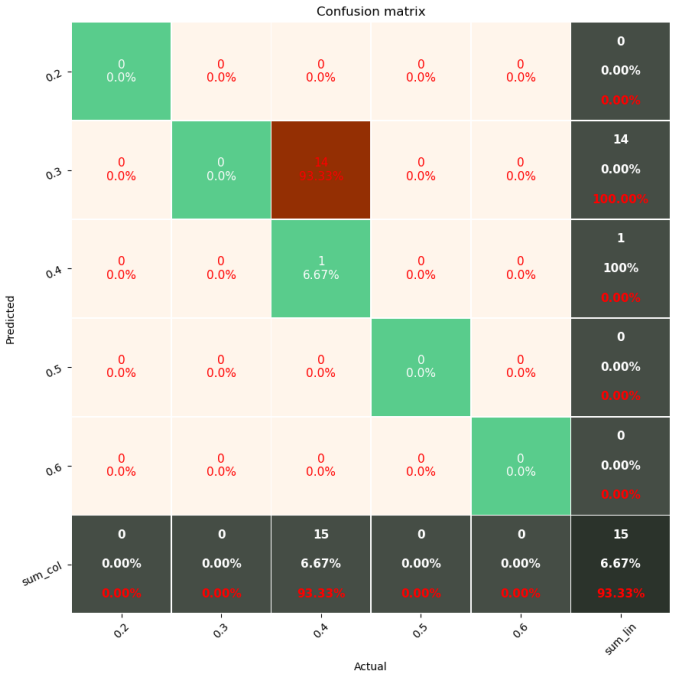
בשלב זה בדקנו את ביצועי המערכת כאשר הערך של ה-T_{60} שהכנסנו לאלגוריתם הוא מהווקטור המשוערך שקיבלנו מרשת הרגרסיה.

הכנסנו את תמונות ה-*STFT* של קטע הקול שבנינו לרשת הרגרסיה המאומנת מחלק ב'.

התוצאות שהתקבלו:



לאחר עיגול התוצאות שהתקבלו:

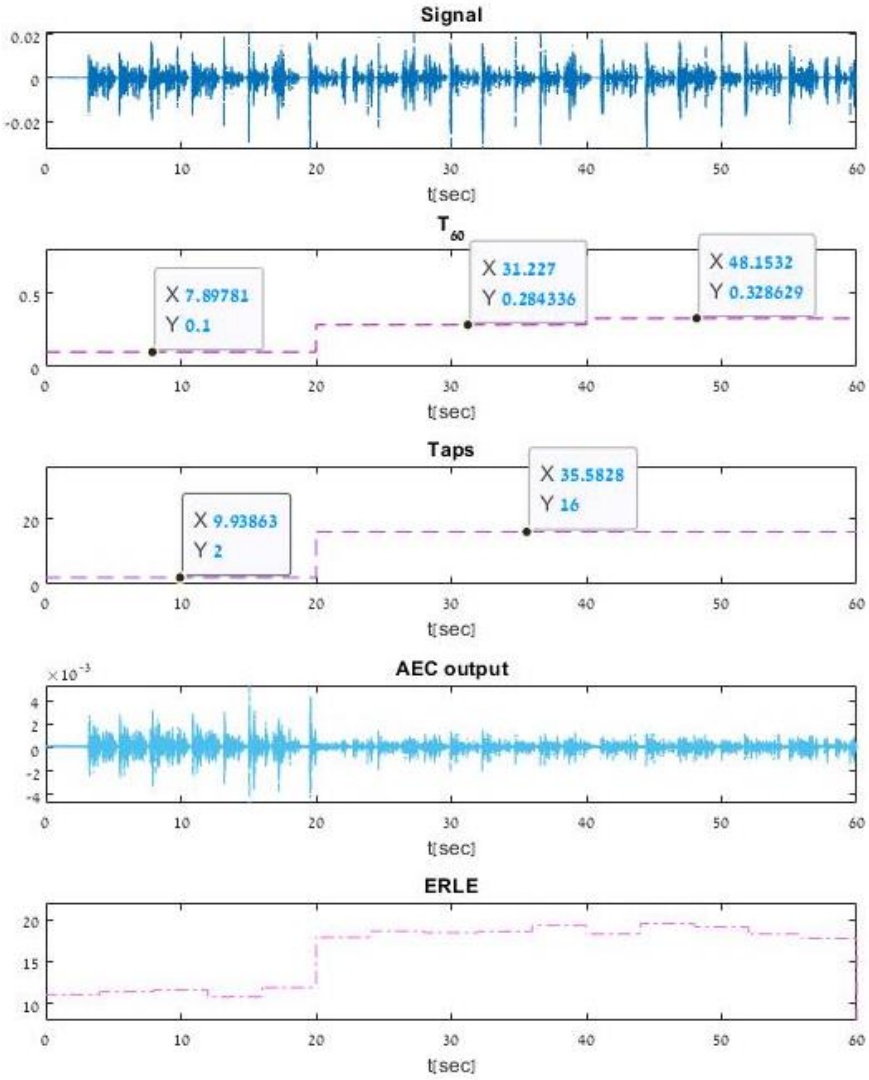


ניתן לראות שהתוצאות שהתקבלו תואמות לתוצאות שקיבלנו בחלק הקודם. יש אחידות בשערוך, פרט לתמונה אחת.
נציין כי במקרה שלנו זמן ההדהוד האמיתי הינו $T_{60} = 0.36[s]$. לכן, במקרה זה הטעות בשערוך נמוכה מ$0.1[s]$.

קיבלנו את הווקטור הבא:

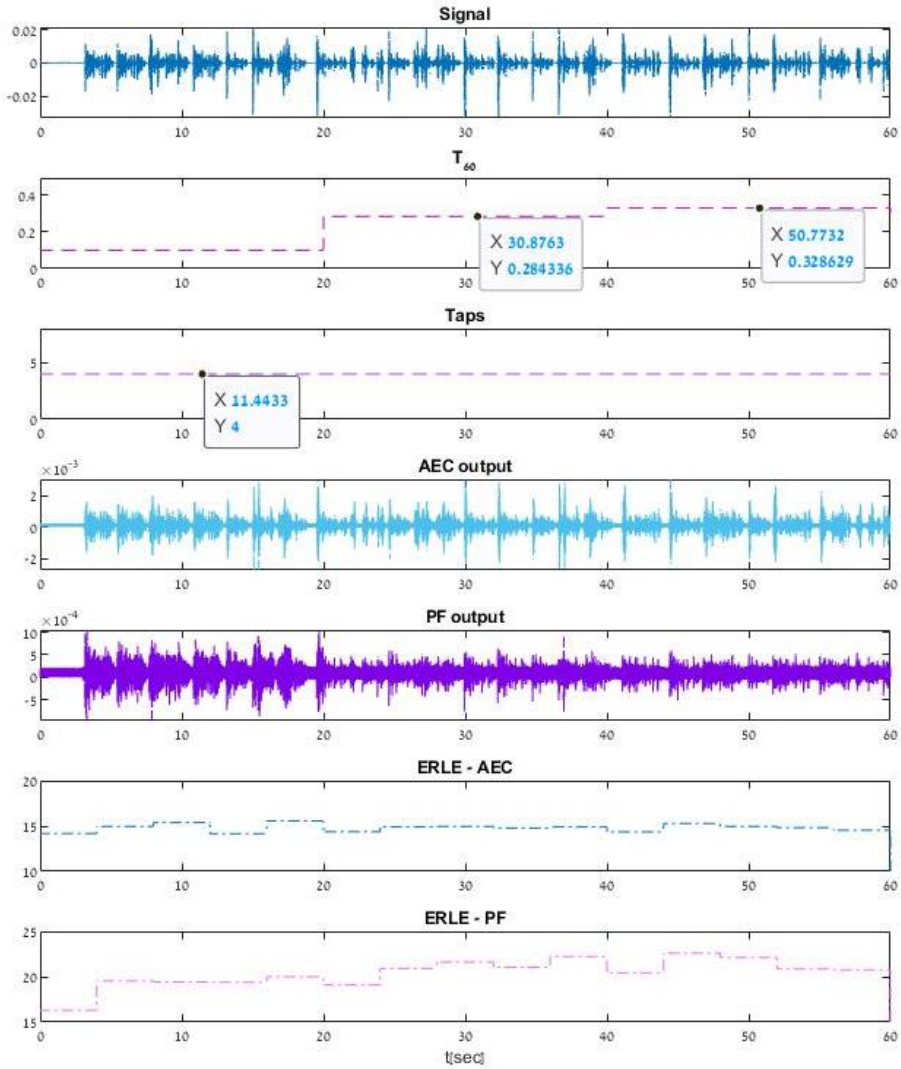
0.275	0.291	0.305	0.297	0.254	0.344	0.322	0.349	0.347	0.281	0.285	0.360	0.292	0.285	0.291

הכנסנו את הווקטור לאלגוריתמים שמימשנו וקיבלנו את התוצאות הבאות:
מוצא ה-AEC:



ניתן לראות שיש שיפור ניכר בביצועים כאשר זמן ההדהוד שאנחנו מכניסים מתאים לזמן ההדהוד במוצא הרשת. נזכור כי אנחנו ממצעים על מספר ערכים לכן, לאחר 20 שניות ראשונות ה-T_{60} היה קצת יותר גבוה. במקרה זה, עבור שני זמני ההדהוד שקיבלנו מספר הטפים היה זהה לכן עבור ה-AEC לא נראה שינוי ניכר בביצועים.

מוצא ה-PF:



ניתן לראות שבצורה דומה למה שקיבלנו קודם, ההנחתה עבור מוצא ה-AEC יחסית קבועה וסביב הערך $15[dB]$.
עבור ה-*PF* ניתן לראות שכאשר זמן ההדהוד נמוך קיבלנו הנחתה נמוכה יותר וכאשר הגדלנו את זמן ההדהוד קיבלנו שההנחתה השתפרה.
נציין כי ברשת זו בכל פעם המוצא היה שונה, לכן, מיצענו על ארבעה ערכים בכל פעם וכך בחרנו את הערך של ה-T_{60} שאותו הכנסנו לאלגוריתם.

רעיונות להמשך

- כדי לשפר את ביצועי האלגוריתמים בחלק א' ניתן להוסיף גלאי סף לזיהוי הדובר הקרוב.

אצלנו ידענו מראש מתי רק הדובר הרחוק מדבר ולמדנו בזמן הזה. לכן כדי לשפר את ביצועי האלגוריתמים בחלק א' ניתן להוסיף גלאי סף לזיהוי הדובר הקרוב. עבור ה-AEC, נפעיל את האלגוריתם של ה-RLS עם מקדם עדכון גבוה יותר-α כאשר רק הדובר הרחוק נוכח וכאשר יש דיבור של שני הדוברים יחד לא נעדכן את מקדמי המסנן בצורה משמעותית.

עבור ה-PostFilter, נקבל מסנן שונה שתלוי בשני המקרים, היה דיבור רצוי או לא.
נקבל:

$$G_{OM-LSA}(l,k) = \{G_{H_1}(l,k)\}^{p(l,k)} \cdot \{G_{H_0}(l,k)\}^{1-p(l,k)}$$

כאשר $p(l,k)$ זה גלאי הסף ההסתברותי.
במקרה כזה נקבל הנחתה נמוכה יותר במקרה שהדובר הקרוב נוכח ולא ננחית את האות הרצוי וכאשר אין דובר רצוי נוכל להנחית בצורה חזקה יותר.

- לשיפור ה-Postfilter ניתן למצוא את הפרמטר $T_{60}(k)$ עבור כל תדר כמו שצוין במאמר. כך נוכל לשפר את ביצועי המסנן.
- לשיפור השערוך, ניתן לממש רשת שונה, רשת RNN. ידוע שרשתות קונבולוציה הן רשתות שיותר מתאימות לניתוח של תמונות. במקרה הזה ניתן לשערך את ה- T_{60} במישור הזמן. רשתות RNN יותר מתאימות למשפטים וקטעי דיבור. לכן, יכול להיות שהביצועים ישתפרו ברשת זו.

מסקנות וסיכום

בפרויקט ביצענו שלושה שלבים שונים:

- **שלב א':** בניית מערכת אקוסטית (AEC) בניית מסנן הדעכת הד שיורי (PF) המשתמשים בזמן ההדהוד לקבלת ביטול הד מיטבי.
- **שלב ב':** בניית מערכת למידה עמוקה לשערוך הפרמטר T_{60}.
- **שלב ג':** חיבור בטור של המערכות הנ"ל ובדיקה כוללת של המערכת.

השלב הראשון עסק באלגוריתמי עיבוד אות והשלב השני בעולם של למידה עמוקה. בשלב השלישי שילבנו בין המערכות וראינו כיצד בעזרת כלים של למידה עמוקה ניתן לשפר את הביצועים.

ביבליוגרפיה

1. Blind reverberation time estimation using a convolutional neural network
2. Habets, E. A. P., Gannot, S., Cohen, I., & Sommen, P. C. W. (2008). Joint dereverberation and residual echo suppression of speech signals in noisy environments. IEEE Transactions on Audio, Speech, and Language Processing, 16(8), 1433-1451.
3. Index of /~gannot/RIR_DATABASE (biu.ac.il)
4. https://www.audiolabs-erlangen.de/fau/professor/habets/software/rir-generator