הפקולטה להנדסה

המעבדה לעיבוד אותות

הפרדת דוברים על ידי שערוך מערכת

רועי זיני

אביעד ברהולץ

פרויקט שנה ד' לקראת תואר ראשון בהנדסה

מנחה: אביעד אייזנברג

מנחה אקדמי: פרופ' שרון גנות

אוקטובר 2022

הצגת המשימה:

במקרים רבים נדרשת יכולת הפרדה של דוברים שונים מתוך מאגר מידע מסוים, בשנים האחרונות חלה עליה בשימוש בשיטות למידה עמוקה (רשתות נוירונים) לצורך פתרון בעיה זו.

בפרויקט זה אנו נשלב בין שיטות קיימות לשיטות למידה עמוקה לצורך פתרון הבעיה כאשר הבעיה איתה נתמודד תהיה הפרדת דיבור של דובר יחיד מתוך הקלטה של דוברים בחדר.

כניסת מערכת ההפרדה תהיה הקלטה מתוך מערך מיקרופונים ומוצאה יהיה הקלטה נקיה של אחד מהדוברים בכניסה.

מערכת ההפרדה שבנינו מבוססת על רשת נוירונים עמוקה שמסתמכת על מאפיינים מרחביים יחסיים, עליהם נסביר בהמשך ספר הפרויקט, שנגזרים מתוך ההקלטות הנקלטות במיקרופונים.

תקציר:

בפרויקט זה התמקדנו בשערוך מערכת, תגובת ההלם של החדר, עבור מספר הדוברים הנמצאים בחדר.

ראשית, בעזרת קוד python יצרנו מערכת הדמיה של חדר המכיל מערך מיקרופונים בעל מספר קבוע של מיקרופונים אשר ישמשו כקלט למערכת ההפרדה ומספר קבוע של דוברים, בקוד עצמו אפשרנו להרחיב למספר דוברים משתנה מסימולציה לסימולציה אך הקוד עצמו לא נבדק, סימולציה זו אחראית לייצור ההקלטה עבור כל מיקרופון המוגדר במערך המיקרופונים הנמצא בחדר.

התוצרים הנשמרים מתוך ההדמיה הינם ההקלטה מתוך מספר המיקרופונים, אותות הדיבור הנקיים שנלקחו מתוך מאגר המידע של המעבדה לעיבוד אותות, הקלטה עבור כל דובר, הRIR, room impulse response, שייצרנו על ידי החבילה שנכתבה על ידי Marvin182 [1] וכיוון ההגעה של הקול למערך המיקרופונים , direction of arrival, doa.

לאחר ייצור ההקלטות חילצנו מתוכן מאפיינים במישור התדר זאת כדי להקל על למידת הרשת.

קלט הרשת היה במישור התדר, לאחר חישוב התמרת פורייה לזמן קצר, stft, על כל אות קול שנקלט חילקנו את תוצאת ההתמרה בתוצאה של מיקרופון ייחוס, כאשר מיקרופון הייחוס הינו המיקרופון הראשון במערך המיקרופונים.

מבנה הרשת בה בחרנו להשתמש הוא ארכיטקטורה של U-NET, רשת קונבולוציה עמוקה, רשת הU-NET בה השתמשנו מורכבת ממקודד ומפענח, כאשר המקודד מבצע embedding למידע המחלחל דרכו והמפענח אמור להביא את המוצא של המערכת לאות הייחוס שלנו, לפי הגדרתינו אות הייחוס של הרשת יהיה המערכת בין הדובר למיקרופונים שייצרנו על ידי החבילה שנכתבה על ידי Marvin182 [1].

במוצא הרשת קיבלנו מימד נוסף שנגזר ממימד הframes של ה-stft, נרצה להיפטר ממנו, את הורדת המימד בחרנו לבצע על ידי שכבה בעלת קישוריות מלאה, fully connected מממד הframes לממד יחיד.

מכיוון שהרשת מקבלת ומוציאה אותות במישור התדר המרנו את הrir שקיבלנו לתגובה במישור התדר ואת האות הזה הכנסנו לרשת כאות המטרה, את המעבר ממישור הזמן לתדר ביצענו על ידי שימוש במסנן וינר.

לאחר אימון המודל ביצענו בדיקה על תוצרי המערכת על ידי ניסיון הפרדה של אותות הדיבור בהקלטה על ידי מסנן mvdr, Minimum Variance Distortion less Response.

[1] https://github.com/Marvin182/rir-generator

רקע תיאורטי:

בחלק זה נסקור מספר מושגי יסוד בהם השתמשנו בתהליך העבודה של הפרויקט.

STFT – Short-time Fourier transform

זוהי טרנספורמציה פורייה עבור זמן קצר (STFT), זוהי התמרת פורייה המשמשת לקביעת התדר הסינוסואידאלי ותכולת הפאזה של מקטעים מקומיים של האות כפי שהוא משתנה לאורך זמן, אנו משתמשים בהתמרה זו מכיוון שהאותות שאנו מתמודדים איתם, אותות הדיבור, אינם סטציונרים כלומר הספקטרום שלהם משתנה בזמן.

חישוב ה-STFT מתבצע על ידי חלוקת האות למקטעים, כאשר אורך החלון של המקטע והחפיפה בין המקטעים נקבעים בהתאם לדרישות, לאחר מכן מתבצעת טרנספורמציה פורייה על כל מקטע בנפרד.

במהלך אימון המערכת התייחסנו לסוג החלון והפרמטרים שלו כאל היפר-פרמטר ששינוי שלו עשוי להניב תוצאות טובות יותר או פחות.

$$X[n,\omega) = \sum_{m=-\infty}^{\infty} w[n-m]x[m]e^{-j\omega m}$$

החלון בו משתמשים יכול להיות סופי או אינסופי במשך הזמן, מקובל להעריך את ה-STFT רק בקבוצה סופית של נקודות ברווח שווה לאורך ציר התדר, בדיוק כפי שה-DFT משמש לעתים קרובות במקום ה-DTFT ביישומים קונבנציונליים של עיבוד אותות דיגיטלי.

נשתמש במשתנה N כדי לציין את מספר ערוצי התדר הבדידים המשמשים ב-STFT, ובמשתנה Nw כדי לציין את אורך פונקציית החלון w[n] כאשר היא מוגבלת למשך זמן פרמטרים אלו מגדירים את הרזולוציה ואת השפעת פסי התדר אחד על השני.

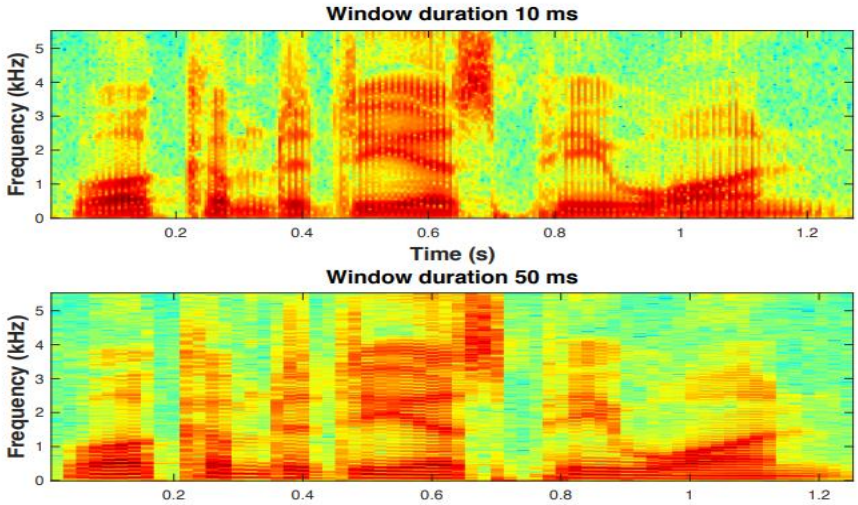
התמרת STFT

$$X[n,k] = \sum_{m=n-(N_w-1)}^{n} w[n-m]x[m]e^{-j\omega_k m} = \sum_{m=n-(N_w-1)}^{n} w[n-m]x[m]e^{-j2\pi mk/N}$$

ניתן לשים לב ש-X[n,k] היא פונקציה של זמן וגם של תדר ועכשיו גם הזמן וגם משתני תדירות הם בדידים.

המשתנה n מציין את מיקומו של חלון הניתוח לאורך ציר הזמן, וקטע הזמן המופרד על ידי החלון. המשתנה k הוא מדד תדירות.

דוגמה להשפעת משך החלון על ה-STFT:



השימוש בחלון ניתוח קצר w[n] יעשה לתת לנו רזולוציה זמנית טובה על חשבון טשטוש רב בתדירות, תוך כדי חלון זמני רחב יותר יספק רזולוציה ספקטרלית חדה על חשבון רזולוציה זמנית מופחתת.

RTF – relative transfer function

מערכי מיקרופונים נמצאים בשימוש נרחב ביישומי הפרדת דיבור.

ניתן לראות את השיפור בכך שבוחרים מספר רמקולים רצויים בסביבה רועשת, תוך שימוש בגישת פונקציית העברה יחסית (RTF).

בשימוש בRTF הוא מייצר אות עם רעש מופחת בהשוואה לאות שהתקבל על ידי מיקרופון (ללא ייחוס) לעומת אות המשויך שהתקבל בשיטת ה-RTF, תוך שמירה על רכיב הדיבור שלו לא מעוות.

מיקרופון הייחוס נבחר בדרך כלל כמיקרופון עם יחס האות לרעש (SNR) הגבוה ביותר. ובעצם מחלקים את כל אותות המיקרופונים באותו המיקרופון.

CNN – convolution neural network

האלמנט הבסיסי ביותר ברשתות קונבולוציה הינו שכבת קונבולוציה, המבצעת קונבולוציה לינארית על פני דאטה בכדי לקבל ייצוג אחר ופשוט יותר שלו.

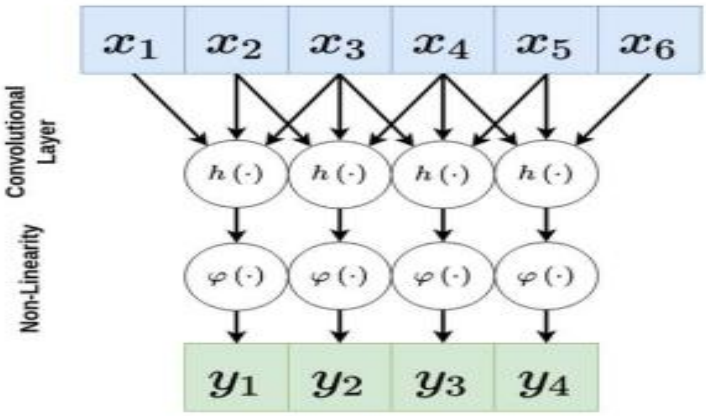
לרוב, שכבת קונבולוציה מבצעת פעולת קרוס-קורלציה בין וקטור המשקלים לבין input מסוים (וקטור הכניסה או וקטור היוצא משכבה חבויה).

וקטור המשקלים נקרא גרעין הקונבולוציה kernel convolution או מסנן filter , בעזרתו מבוצעת פעולת הקרוס-קורלציה.

מלבד הקטנת כמות המשקלים, השימוש בגרעין קונבולוציה מסייע לזיהוי דפוסים ולמציאת מאפיינים.

יכולות אלו נובעות מאופי פעולת הקונבולוציה, הבודקת חפיפה בין חלקים מווקטור הכניסה לבין גרעין הקונבולוציה. הקונבולוציה יכולה למצוא מאפיינים בסיגנל, וישנם גרעיני קונבולוציה שיכולים לבצע אוסף פעולות שימושיות, כמו למשל החלקה ,נגזרת ועוד. אם מטילים על תמונה הרבה גרעינים שונים, ניתן למצוא בה כל מיני מאפיינים – למשל אם הגרעין הוא בצורה של עין או אף, אז הוא מסוגל למצוא את האזורים בתמונה המקורית הדומים לעין או אף.

המוצא של שכבת הקונבולוציה עובר בפונקציית הפעלה לא לינארית (בדרך כלל tanh או, ReLU והוא מכונה מפת הפעלה map activation או מפת מאפיינים. map feature הקונבולוציה יחד עם האקטיבציה נראות כך:



לרוב בכל שכבת קונבולוציה יהיו כמה מסננים, אשר כל אחד מהם אמור ללמוד מאפיין אחר בתמונה. ככל שהרשת הולכת ומעמיקה, כך המאפיינים בתמונה אמורים להיות מובחנים באופן חד יותר אחד מהשני, ולכן המסננים בשכבות

העמוקות אמורים להבדיל בין דברים מורכבים יותר. למשל, פעמים רבות ניתן להבחין כי המסננים בשכבות הראשונות יזהו את שפות האלמנטים שבתמונה או בצורות אבסטרקטיות, ואילו מסננים בשכבות העמוקות יותר יזהו אלמנטים מורכבים יותר כמו איברים או חפצים שלמים בעלי צורה ידועה ומוגדרת .הקלט של שכבת הקונבולוציה יכול להיות רב ערוצי (למשל, תמונה צבעונית המיוצגת לרוב בעזרת ערכי RGB). במקרה זה הקונבולוציה יכולה לבצע פעולה על כל הערוצים יחד ולספק פלט חד ערוצי והיא יכולה גם לבצע פעולה על כל ערוץ בנפרד ובכך לספק פלט רב ערוצי. גרעין הקונבולוציה יכול להיות חד ממדי, כלומר וקטור שפועל על קלט מסוים, אך הוא יכול להיות גם מממד גבוה יותר. לרוב, מסננים הפועלים על תמונות הינם דו ממדיים, ופעולת הקונבלוציה מבצעת בכל שלב כפל בין המסנן לבין אזור דו ממדי אחר בתמונה.

גם ברשת קונבולוציה יש היפר-פרמטרים הנקבעים מראש וקובעים את אופן פעולת הרשת. ישנם שני פרמטרים של שכבת הקונבולוציה – גודל המסנן ומספר ערוצי הקלט וכן שלושה פרמטרים עיקריים נוספים הקובעים את אופן פעולת הקונבולוציה :

ריפוד (Padding) : פעולת הקונבולוציה המוגדרת בעזרת המסנן הינה פעולה מרחבית, כלומר, המסנן פועל על מספר איברים בכל פעולה. בנוסף, נשים לב כי פעולת הקונבולוציה לא מוגדרת על איברי הקצוות לכן לא נוכל להפעיל את המסנן במקומות אלו. אם רוצים לבצע את הקונבולוציה גם על הקצוות, ניתן לרפד את שולי הקלט (באפסים או שכפול של ערכי הקצה).

התרחבות(Dilation) ,על מנת לצמצם עוד במספר החישובים, אפשר לפעול על אזורים יותר גדולים מתוך הנחה שערכים קרובים גיאוגרפית הם בעלי ערך זהה. לשם כך ניתן להרחיב את פעולת הקונבלוציה תוך השמטה של ערכים קרובים. התרחבות טיפוסית הינה בעלת פרמטר $d = 2$.

גודל צעד (Stride) : ניתן להניח שלרוב הקשר המרחבי נשמר באזורים קרובים, לכן על מנת להקטין בחישוביות ניתן לדלג על הפלט ולהפעיל את פעולת הקונבלוציה באופן יותר דליל. כלומר, אין צורך להטיל את המסנן על כל האזורים 3האפשריים ברשת, אלא ניתן לבצע דילוגים, כך שלאחר כל חישוב קונבולוציה יבוצע דילוג בגודל הצעד לפני הקונבולוציה הבאה. גודל צעד טיפוסי הינו $s = 2$. גודל שכבת הפלט לאחר ביצוע הקונבולוציה תלוי בגדלים של הכניסה והמסנן, בריפוד באפסים ובגודל הצעד. מספר שכבות הפלט הינו כמספר המסננים (כאשר שכבת פלט יכולה להיות רב ערוצית). יש לשים לב שערכי ההיפר-פרמטרים (stride, dilation, padding) וכן גודל הגרעין נדרשים להיות מספרים טבעיים.

פעמים רבות דאטה מרחבי מאופיין בכך שאיברים קרובים דומים אחד לשני, למשל – פיקסלים סמוכים לרוב יהיו בעלי אותו ערך. ניתן לנצל עובדה זו בכדי להוריד את מספר החישובים הדרוש בעזרת דילוגים (Strides) או הרחבה (dilation) כפי שתואר לעיל. שיטה אחרת לניצול עובדה זו היא לבצע Pooling – אחרי כל ביצוע קונבולוציה, דגימת ערך יחיד מאזור בעל ערכים מרובים, המייצג את האזור. את צורת חישוב הערך של תוצאת ה pooling-ניתן לבחור בכמה דרכים, כאשר המקובלות הן בחירת האיבר הגדול ביותר באזור שלו (pooling max) או את הממוצע של (average pooling).

כלל, תהליך האימון של רשת קונבולוציה זהה לאימון של רשת, FC כאשר ההבדל היחיד הוא בארכיטקטורה של הרשת. יש לשים לב שהמסננים מופעלים על הרבה אזורים שונים, כאשר המשקלים של המסננים בכל צעד שווים ,ולכן אותם משקלים פועלים על אזורים שונים. לשם הפשטות נניח ויש מסנן יחיד, כלומר מטריצה אחת נלמדת של משקלים. מטריצה זו מוכפלת בכל אחד מהאזורים השונים של הדאטה, וכדי לבצע עדכון למשקלים שלה יש לשקלל את הגרדיאנטים של כל האזורים השונים. בפועל, הגרדיאנט בכל צעד יהיה הסכום של הגרדיאנטים על פני כל הדאטה, ועבור המקרה הכללי בו יש N אזורים שונים עליהם מופעל המסנן הגרדיאנט יהיה:

בדומה ל, FC-גם ב CNN-ניתן לבצע, Normalization Batch-Mini כאשר יש כמה אפשרויות לבצע את הנרמול על סט של וקטורים מסוימים (לשם הנוחות נתייחס לווקטורים של הדאטה כתמונות). אפשרות פשוטה ונפוצה היא לנרמל כל מסנן בפני עצמו על פני כמה תמונות, (Norm Batch) כלומר לקחת את כל הפיקסלים בסט של תמונות ולנרמל בתוחלת ובשונות שלהם. אפשרות נוספת היא לקחת חלק מהמידע של סט תמונות, אך לנרמל אותו ביחס לאותו מידע על פני מסננים אחרים. (Norm Layer) יש וריאציות של הנרמולים האלה, כמו למשל, Norm Instance הלוקח מסנן אחד ותמונה אחת ומנרמל את הפיקסלים של אותה תמונה.

ברשתות קונבולוציה המיועדות לסיווג, בסוף התהליך מתקבל וקטור של הסתברויות, כאשר כל איבר הוא הסתברות של label מסוים. במשימת סגמנטציה זה בעייתי, כיוון שצריך בסוף התהליך לא רק ללמוד את המאפיינים שבתמונה ועל פיהם לקבוע מה יש בתמונה, אלא צריך גם לשחזר את מיקומי הפיקסלים והתיוגים שלהם ביחס לתמונה המקורית עם הסגמנטציה המתאימה.

כדי להתמודד עם בעיה זו הציעו את ארכיטקטורת, Net-U המכילה שלושה חלקים עיקריים: כיווץ, צוואר בקבוק והרחבה (section expansion and, bottleneck, contraction).

בחלק הראשון יש טופולוגיה רגילה של רשת קונבולוציה, המבוצעת בעזרת שכבות קונבולוציה וביצוע pooling השוני בין השלב הזה לבין רשת קונבולוציה קלאסית הוא החיבור שיש בין כל שלב בתהליך לבין חלקים בהמשך התהליך.

לאחר המעבר בצוואר הבקבוק יש למעשה שחזור של התמונה עם הסגמנטציה. השחזור נעשה בעזרת sampling-up על הווקטור שהתקבל במוצא צוואר הבקבוק יחד עם המידע שנשאר מהחלק הראשון של התהליך

PIT – permutation invariant training

רשת הנוירונים PIT מאומנת המערכות הספציפיות לכל דובר, ולאחר מכן על ידי בדיקת כל הפרמוטציות קובעת את הקצאת תווית הפלט הטובה ביותר אשר ממזערת את קריטריון האימון, במקרה שלנו, שגיאת ההפרדה על פי קריטריון מינימום שגיאה ריבועית MSE.

בהפרדת דיבור חד-ערוצי, אנו מניחים שאותות הדיבור עורבבו באופן ליניארי המטרה היא לחלץ את כל אותות הדיבור מהאות המעורב (כלומר, y), לשם כך, אותות מועברים בדרך כלל לתחום התדרים באמצעות טרנספורמציה פורייה קצרת זמן (STFT), מכיוון ש-STFT מספק ייצוג טוב יותר להרמוניות וצפיפות אנרגיה. ברוב מערכות הפרדת הדיבור, החלק של הפאזה מתקבל מהאות המעורבב, מה שהופך את הפרדת הדיבור למשימה של הערכת ספקטרום בגודל של אותות הדיבור.

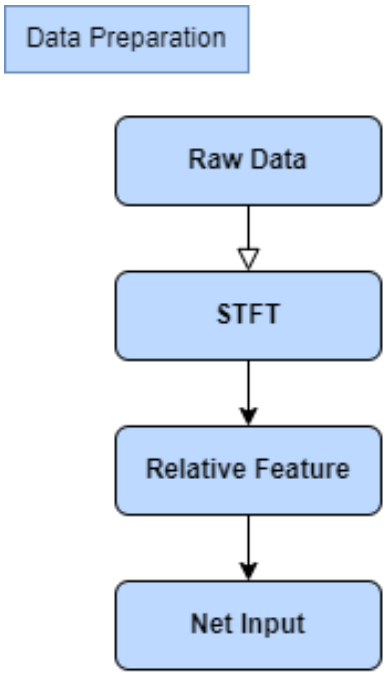
גישה זו זוכה לתשומת לב גוברת בשנים האחרונות. בשל בעיית בבחירת תיוג, איכות המודל משתנה ביחס לנתונים, והפרדת הדיבור היא מאתגרת וצריך לדייק את פונקציית הלוס ויש לבדוק את כל האפשריות.

MVDR Beamformer

MVDR Beamformer הוא פתרון ליצירת beamforming אדפטיבית שמטרתו למזער את השונות של פלט beamformer. אם הרעש והאות הרצוי הבסיסי אינם מתואמים, כפי שקורה בדרך כלל, אז השונות של האותות שנלכדו היא סכום השונות של האות הרצוי ושל הרעש. לפיכך, פתרון ה-MVDR מבקש למזער סכום זה, ובכך להפחית את השפעת הרעש.

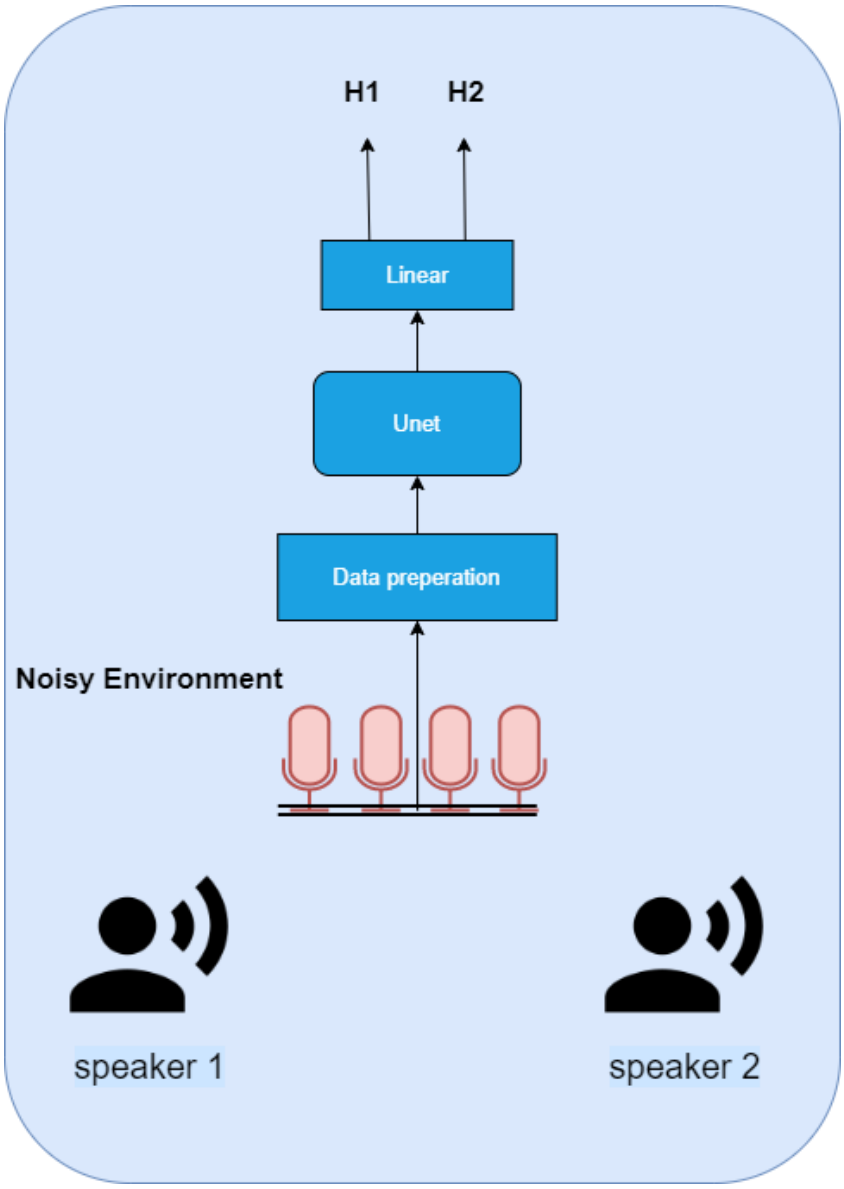
תרשימים:

תהליך עיבוד אותות השמע לפני כניסתם הרשת



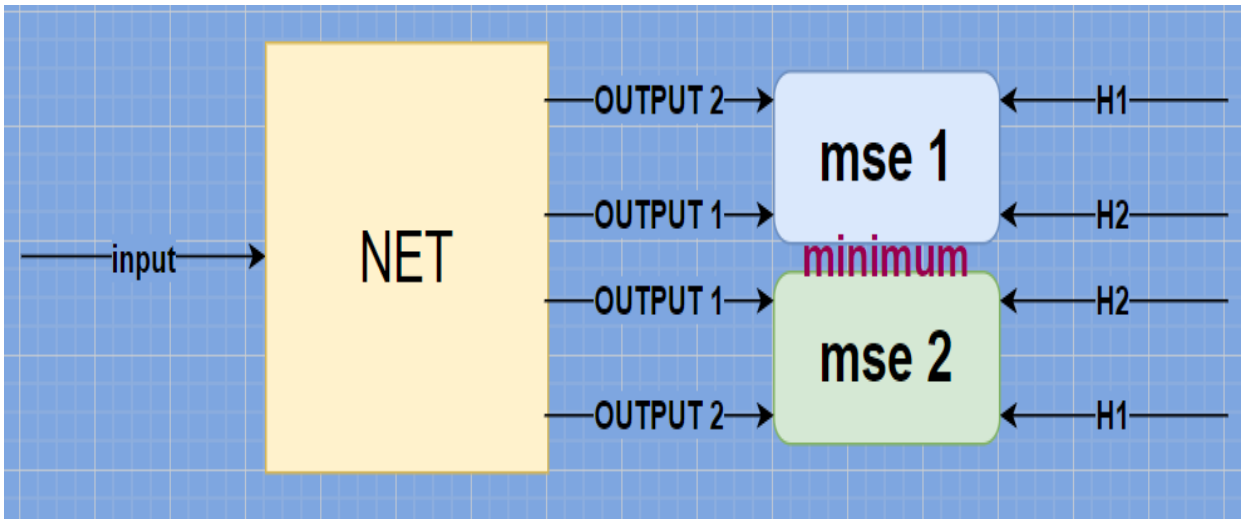
כפי שניתן לראות באיור אותות הקול המיוצרים על ידי הסימולציה עוברים התמרת פורייה לזמן קצר ולאחר מכן בדומה לחישובי RTF אנו מוציאים מתוך האות במישור התדר מאפיין יחסי למיקרופון הראשון, את החישוב הסברנו בהמשך, המאפיין היחסי הוא זה שנכנס לרשת.

תיאור המערכת עד למוצא הרשת



בדיאגרמה הנתונה ניתן לראות כי האותות שייצרנו בסימולציה נכנסים לבלוק הכנת המידע לכניסה לרשת ואת מבנה הרשת בצורה מופשטת.

הסקיצה הבאה מייצגת את הקריטריון לאימון בו השתמשנו



ניתן לראות כי השתמשנו בקריטריון MSE, שגיאה ריבועית, כך שהקריטריון מחושב על פרמוטציות שונות של המידע ופונקציית המחיר שלנו תהיה המינימום בין השניים, במקרה הנתון יש 2 דוברים בחדר.

חישוב הפרמוטציות התבצע על ידי רשת PIT עליה הרחבנו בהמשך.

תיאור השיטה:

תהליך האימון:

השתמשנו במעטפת קונפיגורציה בשם hydra בעזרתה ניתן להריץ את הקוד בעזרת קובצי קונפיגורציה שאחראיים על פרמטרי המודל **[1]**.

המעטפת הזו נתנה לנו גמישות בהגדרת הפרמטרים והמשתנים כך שנוכל לשנות ולבדוק מודלים שונים עם פרמטרים שונים ללא שינוי הקוד עצמו.

הקוד הבא מכיל את מעטפת אימון המודל, כגון, טעינת פרמטרי המודל, הגדרת המודל מתוך מודל pytorch-lightning שהגדרנו בקובץ שונה יחד עם כל תהליך האימון **[2]**, שמירת המודל תוך כדי הריצה ושמירת הגרסה הסופית על ידי פיקוח על מזעור פונקציית המחיר וטעינת המידע הרלוונטי לתוך תהליך האימון תוך עבודה עם מגוון כלים שמקלים על תהליך האימון ומתחשבים ביכולת המחשוב של המחשב עליו אנו מאמנים לדוגמה שימוש במודל טעינת דאטה שבנינו בעזרת מעטפת ה-lightning המאפשר עיבוד מקדים של המידע הנקלט מהחיישנים **[6]**, במקרה שלנו מיקרופונים, בצורה מסודרת ושרשור שלו בקצב שלא "יחנוק" את הזיכרון של הgpu, כמובן שהתייחסנו לפרמטרים נוספים כמו זיכרון ram וזמן ריצה של המודל עליהם נפרט בהמשך בפירוט.

בקוד הבא ניתן לראות כי בעזרת אובייקט הtrainer של המעטפת ה-lightning ,שמגדיר את האספקטים הטכניים של האימון **[5]**, הרצנו את המודל שלנו על מספר gpu דבר שאפשר חיסכון משמעותי בזמן אימון, בנוסף כדי לפקח על פונקציית המחיר של הרשת השתמשנו בחבילה logger של Tensorboard המאפשר מעקב על ידי גרף בזמן אמת של פונקציית המחיר ופרמטרים שנבחרו על ידינו כדי לדאוג שהמודל אכן ישיג את הקריטריונים שהגדרנו ומספק ויזואליזציה המקלה על הסקת מסקנות **[4]**.

נוסף על כך, על ידי אובייקט callbacks הגדרנו למודל מתי להפסיק לאמן את הרשת כך שכאין שיפור בפרמטרי המודל האימון יפסיק ובכך קיבלנו מסקנות על תוצאות האימון במהירות על ידי עצירת האימון **[3]**.

אימון המודל:

[1]
```
@hydra.main(version_base=None, config_path="./conf",
config_name="config")
def main(cfg: Config_class):
    logger = logging.getLogger(__name__)
    logger.info(f"Training with the following
config:\n{OmegaConf.to_yaml(cfg)}")
```

[2]
```
    model = Unet_Model(cfg)
```

```
    checkpoint_dir = os.path.join(cfg.models_dir, 'checkpoints')
    os.makedirs(checkpoint_dir, exist_ok=True)


    logger_dir = os.path.join(cfg.models_dir, 'logs')
    os.makedirs(logger_dir, exist_ok=True)

    checkpoint_callback = ModelCheckpoint(
        dirpath=checkpoint_dir,
        filename='checkpoint_{epoch:02d}-{val_loss:.2f}',
        monitor='val_loss',
        verbose=True,
        save_last=True,
        save_top_k=1,
        mode='min',
    )

    stop_callback = EarlyStopping(
        monitor='val_loss',
        patience=100,
        mode='min',
        check_finite=True,
    )

    callback_list = [checkpoint_callback, RichProgressBar(),
stop_callback]
```

[3]

```
    tb_logger = TensorBoardLogger(save_dir=logger_dir,
name="my_model")
```

[4]

```
    my_strategy = DDPStrategy(find_unused_parameters=False,
static_graph=True)

    trainer = Trainer(
        accelerator="gpu",
        devices=[0, 1],
        max_epochs=cfg.Training.max_epochs,
        enable_model_summary=True,
        callbacks=callback_list,
        logger=tb_logger,
        strategy=my_strategy,
        sync_batchnorm=True,
    )
```

[5]

```
    dm = MyDataModule(data_info=cfg)
    dm.setup(stage="fit")

    trainer.fit(model, datamodule=dm)


if __name__ == '__main__':
    main()
```

[6]

טעינת המידע:

בעזרת מודל dataset של pytorch בנינו את הdataset עם הדאטה שייצרנו:

```
class CustomDataset(Dataset):
    def __init__(self, data_dir, transforms_params,
transform=RTF_mix, target_transform=H_transform):
        self.data_dir = data_dir
        self.files_list = os.listdir(data_dir)
        self.len = len(self.files_list)

        # transforms:
        self.T_params = transforms_params
        self.transform = transform
        self.target_transform = target_transform

        # According to data generation:
        self.mix_index = 0
        self.target_index = 2
        self.fs = transforms_params.fs

        # Initiated cutting:
        self.seconds = transforms_params.used_secs
        self.samples = int(self.seconds * self.fs)  # samples to load
into the model

    def get_len(self):
        return self.len
    def __len__(self):
        return len(self.files_list)

    def __getitem__(self, idx):
        directory = os.path.join(self.data_dir, self.files_list[idx])
        mix, target = self.loader(directory)

        if self.transform:
            net_input = torch.from_numpy(self.transform(mix,
self.T_params))
        if self.target_transform:
            target_input =
torch.from_numpy(self.target_transform(target, self.T_params))

        return net_input.float(), target_input.float()
```

המתודה הבאה טוענת את המידע הרלוונטי מתוך קובץ דחוס של סימולציה:

```
def loader(self, directory):
    # Loading pickle file:
    item = pickle.load(open(directory, 'rb'))
    mix = np.array(item[self.mix_index])  # [mics number x total
samples] -> records time series
    target = np.array(item[self.target_index])  # [speakers number x
mics number x 2048] -> rir in time series
    del item

    # mix handle(mix out shape is [mics number, used samples]):
    length = mix.shape[1]
    min_index = min(length, self.samples)
    mix_slice = mix[:, 0:min_index]

    # numpy:
    mix_out = np.zeros([mix.shape[0], self.samples])
    mix_out[:, 0:min_index] = mix_slice[:, 0:min_index]

    # target handle(target out shape is [speakers number * mics
number, 2048]):
    speakers_num = target.shape[0]
    mics_num = target.shape[1]

    # if ignore first mic:
    ignore_first_mic = True
    if ignore_first_mic:
        target_out = np.zeros([(speakers_num * (mics_num-1)),
target.shape[2]])
        for i in range(speakers_num-1):
            target_out[i * (mics_num-1):((i + 1) * (mics_num-1)), :]
= target[i, 1:, :]
    else:
        # numpy:
        target_out = np.zeros([speakers_num * mics_num,
target.shape[2]])
        for i in range(speakers_num):
            target_out[i * mics_num:((i + 1) * mics_num), :] =
target[i, :, :]

    return mix_out, target_out
```

הגדרנו את סט האימון, סט הווליד‌ציה וסט הטסט על ידי דאטה מודל של מעטפת ה-lightning של pytorch בצורה הבאה:

```
class MyDataModule(LightningDataModule):
    def __init__(self, data_info):
        super().__init__()
        self.data_dir = data_info.db.train_data_dir
        self.test_data_dir = data_info.db.test_data_dir
        self.T_params = data_info.Data_Transforms
        self.train_ds = None
        self.val_ds = None
        self.test_ds = None

        # dataloaders info:
        self.batch_size = data_info.Training.batch_size

    def setup(self, stage: str = None):
        # Assign train/val datasets for use in dataloaders
        if stage == "fit" or stage is None:
            full_dataset = CustomDataset(data_dir=self.data_dir,
transforms_params=self.T_params)
            n_val = int(full_dataset.get_len() * 0.1)
            n_train = full_dataset.get_len() - n_val
            self.train_ds, self.val_ds = random_split(full_dataset,
[n_train, n_val])

        # Assign test dataset for use in dataloader(s)
        if stage == "test" or stage is None:
            self.test_ds = CustomDataset(data_dir=self.test_data_dir,
transforms_params=self.T_params)

    def train_dataloader(self):
        return DataLoader(self.train_ds, batch_size=self.batch_size,
pin_memory=True, shuffle=True, num_workers=40)

    def val_dataloader(self):
        return DataLoader(self.val_ds, batch_size=self.batch_size,
pin_memory=True, shuffle=False, num_workers=40)

    def test_dataloader(self):
        return DataLoader(self.test_ds, batch_size=self.batch_size,
pin_memory=True, shuffle=False, num_workers=40)
```

השתמשנו במודל הpytorch שכבר הגדרנו על מנת להגדיר את המודל הנ"ל בו בנוסף לחלוקת הדאטה סט המלא ,לסט של אימון ולסט של ווליד‌ציה, הגדרנו את ממד הbatch ומספר המעבדים המעורבים בטעינת המידע ועיבודו ופרמטרים נוספים המאפשרים אקראיות נוספת ומאפשרים שליטה על אופן טעינת המידע לזיכרון.

שלב עיבוד המידע

עיבוד האותות התבצע על המידע בכניסת הרשת אותות השמע ואות המטרה שהגדרנו עבור המערכת.

בשלב הראשון חילצנו מתוך אותות הכניסה את האות הmix שמתוכו אנו שואפים לסנן אות של דובר יחיד.

לאחר החילוץ מתוך הקבצים הדחוסים, העברנו אותות השמע בהתמרת stft.

עבור אותות השמע בכניסת הרשת חישבנו את rtf_mix.

פונקציה זו מקבלת את אותות השמע לאחר חישוב stft ובוחרת מיקרופון יחסי, שיהיה הרפרנס לכל השאר, ואנו מחלקים כל תוצאת stft מיקרופון יחיד בתוצאת מיקרופון הייחוס, מוצא הפונקציה יהיה שרשור של 3 המיקרופון (מיקרופון הייחוס הוא חסר משמעות, חילוק בעצמו יניב מטריצה שאיבריה 1).

כך קיבלנו בסך הכל 3 מקרופונים כאשר כל מיקרופון מחזיק stft יחסי שהוא במימד של *nfft* תדרים על מימד ה-frames, וכל אחד מורכב ממספר מרוכב (הגדרנו כשני ערוצים), בקוד מפורטים המימדים שלהם נצפה במוצא ובכניסה.

```
def RTF_mix(mix, T_params=None):
    """ Relative Transfer Function
        Received data is numpy array in shape->(mics num, samples)
        RTF output is [channel,stft dim 1->according to nfft,stft dim
2->frames num]"""

    eps = 10**-3

    mics_num = mix.shape[0]  # mics num

    # mix handle:
    stft_mix = []
    for i in range(mics_num):
        sig = mix[i, :]  # i index is mic number

        # return a one-sided spectrum for real data, but for complex
data, a two-sided spectrum is always returned:
        stft_calc = signal.stft(sig, nperseg=T_params.nperseg,
window=T_params.window,
                                nfft=T_params.nfft)[2]  # might be
better with other params , noverlap=T_params.noverlap
        stft_mix.append(stft_calc)

    stft_shape = stft_mix[0].shape
    channels_num = (mics_num-1)*2

    rtf = np.zeros([channels_num, stft_shape[0], stft_shape[1]])
    ref_mic_index = 0
```

```
    for i in range(mics_num-1):
        temp = stft_mix[i + 1] / (stft_mix[ref_mic_index] + eps)
        rtf[2*i, :, :] = temp.real
        rtf[2*i+1, :, :] = temp.imag
    return rtf  # numpy array
```

אות המטרה שלנו הוא המערכת ששמרנו בזמן ייצור הדאטה בתדר, כדי להמיר את דגימות מערכת הrir לתגובת המערכת בתדר השתמשנו במסנן וינר כאשר אות הייחוס בחישוב הוא רעש רנדומלי שייצרנו.

```
def PSD(x, y, T_params):
    """ Calculating the power of x,y inputs co-spectrum density using
welch method """
    _, Pxy = signal.csd(x, y, fs=T_params.fs, nfft=T_params.nfft)
    return Pxy


def H_transform(h, T_params):
    """ Winer filter """
    eps = 10 ** -3
    H_out = np.zeros([h.shape[0] * 2, int(T_params.nfft/2 + 1)])
    x = np.random.randn(h.shape[0], h.shape[1])
    for i in range(h.shape[0]):
        y = signal.convolve(h[i, :], x[i, :])[:x.shape[-1]]  # full
discrete linear convolution of the inputs.
        H_temp = PSD(x[i, :], y, T_params)/(PSD(y, y, T_params)+eps)

        H_out[2*i, :] = H_temp.real
        H_out[2*i+1, :] = H_temp.imag

    return H_out
```

הגדרת המודל:

השתמשנו בחבילה בשם **Pytorch lightning** , המאפשרת להגדיר כל שלב בתהליך האימון של רשת הנוירונים.

הגדרנו את מספר הערוצים של הכניסה שהוא 12 ואילו ביציאה הוא 6, עבור כל מיקרופון (3) יש מערכת, H, מספר מרוכב וממשי בסך הכל 6.

```
#  Net definition:
class Unet_Model(pl.LightningModule):
    def __init__(self, hparams):
        super().__init__()
        self.save_hyperparameters(hparams)
        self.train_hp = hparams.Training
        self.net_hp = hparams.Net_hp

        self.in_channels = self.net_hp.n_channels
        self.out_channels = self.net_hp.n_classes

        # self.Unet = Unet(self.in_channels, self.out_channels,
out_sz=2048)
        self.Unet = UNet_v3(self.in_channels, self.out_channels)

        self.lr = self.train_hp.lr
        self.batch_size = self.train_hp.batch_size
        self.train_data_dir = self.hparams.db.train_data_dir

        self.pit_indexes = self.set_pit_indexes()

    def forward(self, x):
        return self.Unet(x)

    def training_step(self, batch):
        x, y = batch
        y_hat = self.Unet(x)

        loss = self.metric(torch.squeeze(y_hat), torch.squeeze(y))

        tensorboard_logs = {'train_loss': loss}
        self.log("loss", loss, on_step=True, on_epoch=True,
prog_bar=True, logger=True, sync_dist=True)
        return {'loss': loss, 'log': tensorboard_logs}
```

הגדרנו את פונקציית הלוס שנעשתה בעזרת ה**PIT** , אשר כל פעם בוחר את השגיאה המינימלית של המודל.

מכיוון שלא ניתן לדעת איזה ערוצים מתאימים לאיזה דובר, אנו בודקים כל פעם איזה שגיאה ריבועית היא יותר קטנה, ועל פיו אני אנו מעדכנים את הרשת, כלומר אנו משווים את 3 הערוצים הראשונים של הטרגט ל3 ערוצים של היציאה מהמערכת, ואז מצליבים, ההצלבה מתבצעת על ידי pit_indexes.

```
def metric(self, y_hat, y):
    #  PIT metric:
    loss_1 = self.my_mse(y_hat, y)
    loss_2 = self.my_mse(y_hat, y[:, self.pit_indexes, :])
    return min(loss_1, loss_2)

@staticmethod
def my_mse(y_hat, y):
    mse = F.mse_loss(y_hat, y)
    return mse
```

במהלך האימון ניסנו מספר שיטות לאופטימיזציה

```
def configure_optimizers(self):
    # default optimizer->adam optimizer:
    if self.train_hp.optimizer == 'adam' or self.train_hp.optimizer
!= 'sgd':
        optimizer = torch.optim.Adam(self.parameters(), lr=self.lr)
    # sgd optimizer:
    if self.train_hp.optimizer == 'sgd':
        optimizer = torch.optim.SGD(self.parameters(), lr=self.lr)
    return {'optimizer': optimizer}
```

מבנה רשת הUNET:

השתמשנו במבנה הunet קלאסי שבדרך כלל משמש למודל של תמונה.

```
class UNet_v3(nn.Module):
    def __init__(self, n_channels, n_classes, bi_linear=False):
        super().__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bi_linear = bi_linear

        # layers def:
        self.inc = DoubleConv(n_channels, 64)
        self.down1 = Down(64, 128)
        self.down2 = Down(128, 256)
        self.down3 = Down(256, 512)
        factor = 2 if bi_linear else 1
        self.down4 = Down(512, 1024 // factor)
        self.up1 = Up(1024, 512 // factor, bi_linear)
        self.up2 = Up(512, 256 // factor, bi_linear)
        self.up3 = Up(256, 128 // factor, bi_linear)
        self.up4 = Up(128, 64, bi_linear)
        self.outc = OutConv(64, n_classes)
        self.linear = nn.Linear(189, 1)

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        x = self.outc(x)
        logits = self.linear(x)

        return torch.squeeze(logits)
```

כשאר ניתן לראות בבירור את המעבר של הערוצים בכל בלוק ובלוק של הרשת:

16->64->128->256->512->1024->512->256->64->16->6

כל בלוק קונבלוציה בסיסי הוגדר עם doubleconv שכבת batch ואז relu :

```
class DoubleConv(nn.Module):
    """ (convolution => [BN] => ReLU) * 2 """

    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3,
padding=1, bias=False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3,
padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)
```

כל up מוגדר בצורה הבאה כבלוק בסיסי ונוסף החיבור של הdown המקביל שלו על מנת לשמור על מאפיינים:

```
class Up(nn.Module):
    """Up scaling then double conv"""

    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()

        # if bi linear, use the normal convolutions to reduce the
number of channels
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear',
align_corners=True)
            self.conv = DoubleConv(in_channels, out_channels,
in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels, in_channels //
2, kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        # input is CHW
        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]

        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)
```

והdown הוגדר בצורה הבאה עם **doubleconv ו maxpool**:

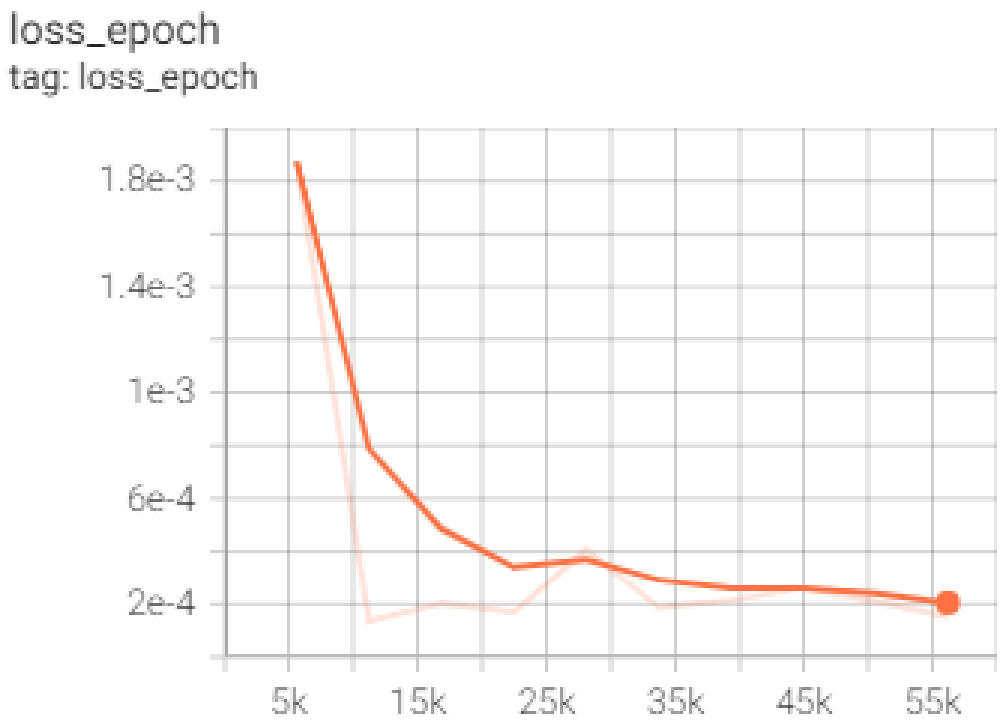
```
class Down(nn.Module):
    """Downscaling with max pool then double conv"""

    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)
```

בנוסף הוספנו שכבה לינארית נוספת אשר ממפה את מימד הזמן שהוא 189, ל1, מכיוון שהטרגט שלנו צריך להיות קבוע בזמן. וכך ניתן להגיע למימד הרצוי של [6*256] כמו של הטרגט.

תוצאות

לאחר שקיבלנו התכנסות של המודל בולדיציה כמו שניתן לראות:

loss_epoch



עברנו לשלב בדיקת המודל על סט הדאטה.

על מנת לבדוק את התוצרים שלנו, השתמשנו בשיטת MVDR .

לקחנו את סט הh שלנו שנוצרו במישור הstft, חילקנו בנורמה של עצמו, ואז הטלנו את המיקס במישור הstft, כלומר הכפלנו בצמוד ההופכי שלו. את התוצאה התמרנו חזרה למישור הזמן וייצרנו קובץ wav שאותו השמענו על מנת לבדוק את ההפרדה.

מסקנות וסיכום

במהלך הפרויקט יצרנו סימולציה המדמה חדר בו יש מערך מיקרופונים ואנשים המדברים בחפיפה זה לזה שמרנו את המערכת בין דובר למיקרופון, אותות הדיבור המקוריים ואת הקלטת המיקרופונים מתוך הסימולציה.

בעזרת המידע שייצרנו אימנו רשת נוירונים עמוקה אשר מטרתה היה בהינתן הקלטה של בליל הדוברים למצוא את המערכת בין דובר לכל אחד מהמיקרופונים ובעזרת מערכות אלו לבצע הפרדה של אותות הדיבור כלומר מוצאה של המערכת צריך להיות אות של דובר יחיד מתוך ההקלטה בכניסה למערכת.

מבדיקה לפי קריטריון מינימום שגיאה מרובעת קיבלנו כי מוצא הרשת עומד בדרישות, כלומר אכן שיערוך המערכת עבד ,אך ,המוצא הסופי של מערכת ההפרדה לאחר שיערוך האות של דובר יחיד עם משערך הmvdr היה קובץ שמע מאוד דומה לקובץ המקורי, כלומר לא התבצעה הפרדה.

לדעתינו בעיית ההפרדה עצמה לא פעלה בגלל שיערוך הMVDR.

במהלך הפרויקט השתמשנו בכלים רבים בעיבוד אותות כגון מסנן וינר לשערוך ספקטורם, התמרה לזמן קצר ששימשה אותנו לעיבוד אותות דיבור שאינם סטציונרים.

נוסף על כך, השתמשנו בכלים תכנותיים כדי לייצר סימולציה, לבנות את הרשת ולהריץ מודל תוך התעסקות עם כלים לניהול קבצי קונפיגורציה, תיקיות ועבודה עם מבנים של פרויקטים בתעשייה כגון עבודה עם Cookiecutter וGitHub,

בנוסף כדי שנוכל לקבל תוצאות בזמן סביר ולוודא שאנחנו עובדים בצורה נכונה התעסקנו עם ניהול משאבים של המחשבים עליהם עבדנו, שיפור זמני ריצה וקומפילציה של הקוד.