הפקולטה להנדסה

המעבדה לעיבוד אותות

Multi-Source Diffusion Models for Simultaneous Music Generation

מעיין חסון

פרויקט שנה ד' לקראת תואר ראשון בהנדסה

מנחה: רננה אופוצ'ינסקי

מנחה אקדמי: פרופ' שרון גנות

אוקטובר 2024

תוכן עניינים

הצגת הבעיה........ 3
רקע תיאורטי 4
AI (Artificial Intelligence)- בינה מלאכותית........ 4
ML (Machine Learning)- למידת מכונה 4
DL (Deep Learning)- למידה עמוקה 5
NN (Neural Network)- רשת נוירונים 6
CNN (Convolutional Neural Network)- רשת קונבולוציה........ 11
U-net 14
Diffusion model- מודל דיפוזיה........ 15
מקורות מוזיקה........ 17
חלופות לפתרון........ 17
שיטת הפתרון 18
פונקציית ה-score........ 19
מודל הדיפוזיה........ 19
גנרציה והפרדה........ 24
אופן פעולת המודל 25
אלגוריתם........ 28
הניסוי........ 30
dataset 30
ארכיטקטורה 30
אמצעי החקר........ 31
אופן השימוש........ 31
הממשק........ 32
מהלך הניסוי........ 32
שיטות שיפור התוצאות........ 33
תוצאות........ 39
תוצאות evaluation עם משקלים מקוריים 41
שינויי פונקציית Loss 44
שינוי אופטימייזר מADAM ל- ADAMW........ 46
התמודדות המודל עם דאטא חדש........ 46
התמודדות המודל עם שימוש חלקי בדאטא 48
אופן השימוש בקלט........ 49
סיכום ומסקנות 50
נספחים........ 51

תקציר

הפרויקט מבוסס על המאמר: "Multi-Source Diffusion Models for Simultaneous Music Generation And Separation".[1] במאמר מוגדרים מודלים גנרטיביים מבוססי דיפוזיה, שביכולתם לסנתז מוזיקה ולהפריד מקורות (דוברים, כלי נגינה). אנו התמקדנו בהפרדת כלי נגינה. מודלי הדיפוזיה מבוססים על למידת פונקציית ה-score של צפיפות ההסתברות המשותפת של מקורות בעלי קשר משותף. בבסיס המודל ישנה U-net אשר מבצעת בפועל את הלמידה. "Dirac likelihood function" היא פונקציית המפתח עבור משימת הפרדת הכלים.

מטרתנו בפרויקט היא להגיע להפרדה המתוארת במאמר, ולנסות לשפר את איכות הפרדת הכלים כפי שהיא מבוצעת במאמר. המאמר מציג הפרדה יחסית טובה, כאשר מדד איכות ההפרדה הוא SI-SDR. ההפרדה מתבצעת באמצעות הכנסת mix כלים לרשת, וקבלת כלים מופרדים במוצא הרשת. שיטת השיפור המרכזית שלנו שינוי פונקציית ה-loss באימון המודל. בשיטה זו התקבל שיפור ניכר.

[1] https://openreview.net/pdf?id=h922Qhkmx1

הצגת הבעיה

בידינו N מקורות מובחנים $\{x_1, \dots, x_N\}$, $\forall n : x_n \in R^D$. המקורות נסכמים זה עם זה בצורה קוהרנטית לקבלת mix:

$$\mathbf{y} = \sum_{n=1}^{N} \mathbf{x}_n$$

ישנן 3 מטרות המוצגות במאמר, כאשר אנו עסקנו בפתרון הבעיה האחרונה:

1. גנרציה כללית: דגימת כל מקור ($\{\mathbf{x}_1, \dots, \mathbf{x}_\mathrm{N}\}$) מתוך ההתפלגות שלו, וסכימת כל הדגימות לקבלת mix חדש.
2. גנרציה חלקית: בהינתן mix חלקי (של חלק מהכלים) $\mathbf{x}_\mathfrak{I}$, מעוניינים לקבל mix של שאר הכלים $\mathbf{x}_{\bar{\mathfrak{I}}}$ ע"י דגימת ההתפלגות המותנית $p(\mathbf{x}_{\bar{\mathfrak{I}}}|\mathbf{x}_\mathfrak{I})$, כך שתהיה הרמוניה בין כל הכלים.
3. הפרדת מקורות (source separation): בהינתן ה-mix $\mathbf{y}$, מעוניינים לבודד את המקורות המרכיבים אותו, באמצעות דגימת ההתפלגות הפוסטריורית $p(\mathbf{x}|\mathbf{y})$.

רקע תיאורטי

AI (Artificial Intelligence)- בינה מלאכותית

בינה מלאכותית הינה תחום העוסק ביצירת מערכות המסוגלות לבצע משימות, אשר בדרך כלל דורשות אינטליגנציה אנושית, כגון זיהוי חזותי, סיווג, תרגום שפה, יצירת טקסט, חיזוי, קבלת החלטות בזמן אמת ועוד. מערכות אלו מבוססות מכונה, אשר מבצעת את המשימה הדרושה בהסתמך על הקלט שהיא מקבלת.

ML (Machine Learning)- למידת מכונה

למידת מכונה הינה תת-תחום של הבינה המלאכותית, הכולל קבוצת אלגוריתמים סטטיסטיים שמטרתם לבצע למידה מתוך דוגמאות. בשונה מתכנות קלאסי, המחשב מבצע את הלמידה מבלי להיות מתוכנת באופן מפורש עבור המשימה הנדרשת.

הלמידה מבוססת על מספר גדול של דוגמאות שהמכונה מקבלת. ככל שהדוגמאות רבות ומגוונות יותר, כך המודל שיתקבל יהיה מוכלל יותר עבור המשימה הנדרשת, ויוכל להתמודד טוב יותר עם המשימה.

דוגמאות לאלגוריתמים נפוצים בתחום זה:

Linera Regression, Logistic Regression, PCA, k-means, LDA, Nearest Neighbor, t-SNE, SVM, HMM, Neural Network.

מקובל לחלק את התחום ל-3 סוגים:

- supervised learning (למידה מונחית): בלמידה זו הדוגמאות שהמכונה מקבלת לשם למידה הן בעלות סיווג. בדרך כלל מטרת למידה כזו תהיה יכולת לבצע סיווג של דוגמאות שהמכונה לא "מכירה" (כלומר, שלא אומנה עליהן). יכולת הסיווג תתבצע מתוך למידת מיפוי בין דוגמאות לסיווג שלהן. רשת נוירונים (Neural Network) משתמשת בגישה זו לצורך הלמידה.

- unsupervised learning (למידה בלתי מונחית): בלמידה זו הדוגמאות אינן בעלות סיווג. בדרך כלל מטרת למידה כזו תהיה למצוא תבניות, קשרים ומבנים עבור אוסף הנתונים. דוגמאות לאלגוריתמים מסוג זה הן k-means, PCA.
- reinforcement learning (למידה מבוססת חיזוקים): בלמידה כזו מטרת המכונה היא ללמוד לקבל החלטות בצורה טובה, כך שהרווח במשימה יהיה מקסימלי. האלגוריתם מקבל משוב ("תגמול" או "עונש") עבור כל החלטה שהוא מקבל, וכך הוא לומד כיצד להחליט נכון. יישומים לדוגמא בהם דרושות החלטות: משחק, רובוט, רכב אוטונומי, מסחר וניהול משאבים.

DL (Deep Learning) -למידה עמוקה

למידה עמוקה היא תת תחום של למידת מכונה, אשר מבוסס על שימוש ברשתות נוירונים בעלות שכבות מרובות ("עמוקות"), שמטרתן ללמוד דפוסים ותכונות ברמות שונות של אוסף נתונים. שכבות ברמת עומק שונות "אחראיות" על תכונות ברמות הפשטה שונות. מודלים בתחום זה מתמודדים בהצלחה עם עיבוד נתונים ממימדים גדולים כגון תמונות, וידאו, אודיו וטקסט.[2]

יתרון של תחום זה הוא שלעיתים קרובות, לאחר קבלת מודל עבור בעיה מסוימת, ניתן להשתמש באותו מודל עבור בעיה אחרת (בעלת קשר מסוים לבעיה המקורית). כלומר, ידע שנרכש עבור משימה אחת, יכול להיות מיושם עבור משימה אחרת. גישה זו נקראת Transfer Learning. דוגמא פשוטה לשימוש כזה היא לאמן רשת לזיהוי כלבים, ולאחר מכן להתאים את הרשת הזו לזיהוי חתולים.

המודלים מסתמכים על מערכי נתונים גדולים, והם דורשים כוח חישוב רב על מנת לקבל ביצועים גבוהים. אימון שכזה דורש לרוב חומרת GPU.

[2] Deep learning course, Stanford: https://cs231n.github.io/

מספר ארכיטקטורות מרכזיות בתחום הן:

- CNN (Convolutional Neural Network)
- RNN (Recurrent Neural Network)
- LSTM (Long Short-Term Memory)
- GANs (Generative Adversarial Network)

הארכיטקטורות השונות מותאמות למשימות שונות. רשתות CNN מותאמות לעיבוד תמונות. רשתות RNN מותאמות לעיבוד מידע טורי כגון טקסט. רשתות LSTM גם הן מותאמות לעיבוד מידע טורי. GANs הם מודלים שמטרתם ביצוע גנרציה של דאטה חדש.

NN (Neural Network)- רשת נוירונים

רשת נוירונים הינה מודל מתמטי חישובי, המכיל מספר רב של צמתים ("נוירונים") המקושרים זה לזה. ברשת ישנן שכבות שונות, כאשר בכל שכבה ישנם מספר נוירונים. כל נוירון מקבל בקלט את ערכי כל הנוירונים שבשכבה הקודמת. על ערכים אלו הוא מופעלת פונקציה לינארית (משקול), ועל התוצאה מופעלת פונקציה לא לינארית- פונקציית אקטיבציה. אי הלינאריות של פונקציות האקטיבציה הוא קריטי. בהיעדרן, הרשת, ללא תלות בעומקה, תתנהג כמו מודל לינארי חד שכבתי (משום שפעולה לינארית על פעולה לינארית נחשבת כפעולה לינארית).

ניתן לבטא את הפלט של נוירון כך (איור 1):

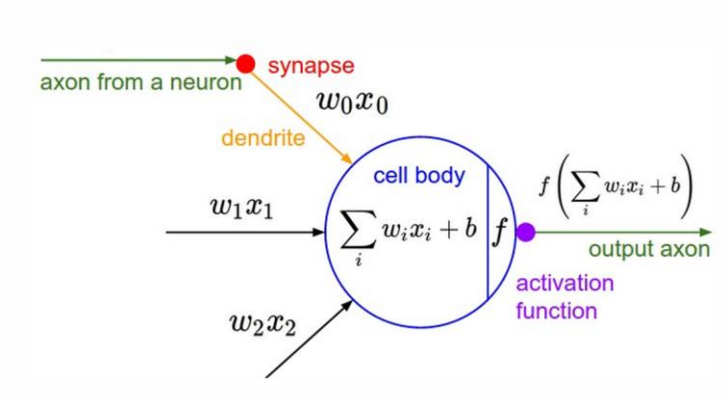
$$\underbrace{f}_{\substack{activation\\function}}\left(\sum_i \underbrace{w_i}_{\text{משקל}} \cdot x_i + \underbrace{b}_{bias}\right)$$

כאשר $\{x_i\}$ הם הנוירונים שבשכבה הקודמת.

נשים לב כי מעבר בין 2 שכבות, כאשר בכל שכבה מספר נוירונים, בעצם מתבצע באופן הבא:

$$\underbrace{\bar{h}_{i+1}}_{\substack{next\\layer}} = \underbrace{A}_{\substack{weights\\matix}} \cdot \underbrace{\bar{h}_i}_{\substack{current\\layer}} + \underbrace{\bar{b}_i}_{bias}$$

כאשר הפעולה העיקרית היא ההכפלה במטריצת המשקלים.



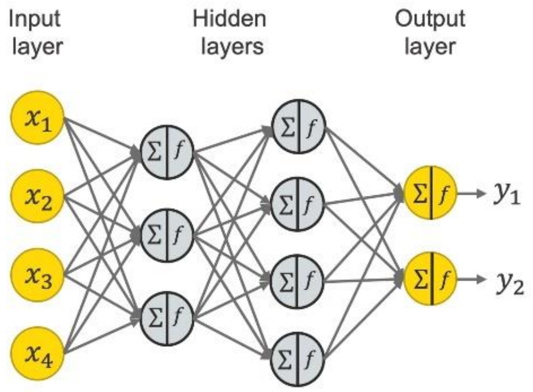
איור 1: נוירון. קומבינציה לינארית של קלטים מועברת כארגומנט לפונקציית אקטיבציה.

פונקציות אקטיבציה נפוצות: ReLU (rectified linear unit), Sigmoid, Tanh, SoftMax.

לפי משפט הקירוב האוניברסלי, רשת נוירונים דו שכבתית העושה שימוש בפונקציית אקטיבציה לא לינארית, יכולה לקרב כל פונקציה.

באופן פשוט, ניתן להתייחס לרשת כאל פונקציה, אשר מופעלת על קלט לקבלת פלט. עבור משימות מורכבות כגון זיהוי, סיווג, גנרציה, ושאר המשימות איתן מנסה להתמודד הבינה המלאכותית, נדרשות פונקציות לא לינאריות מורכבות. לשם כך, עבור כל משימה, דרושה כמות גדולה של נוירונים, שכבות, ופונקציות לא לינאריות (שפועלות באופן טורי- אחת על גבי פלט האחרת). המודל בעצם מבצע סדרה של טרנספורמציות לא לינארית על הקלט, כך שהקשרים הנוצרים מורכבים מאוד.

שכבת הנוירונים הראשונה נחשבת כ-input. שכבת הנוירונים האחרונה נחשבת כ-output, והשכבות שביניהן נחשבות כ-hidden layers. תצורת הרשת היא לרוב fully connected עם Feed Forward (זרימת המידע היא בכיוון אחד- קדימה, ללא לולאות או מעגלים) כמתואר באיור 2.



איור 2: רשת נוירונים. הארכיטקטורה מכילה מספר שכבות המורכבות מנוירונים.

ארכיטקטורה זו התקבלה מתוך השראה ממבנה המוח האנושי. המודל מחקה את הנוירונים במוח, המחוברים ביניהם באמצעות סינפסות. כל נוירון במוח מקבל אותות קלט מנוירונים אחרים, מעבד אותם, ושולח אותות פלט.

על מנת להשתמש ברשת עבור ביצוע המשימות (סיווג וכו') בצורה טובה, יש להגיע למצב בו הפרמטרים של הרשת מכוונים בצורה טובה.

פרמטרי הרשת (Hyperparameters) הם הערכים הניתנים לשינוי, בהינתן רשת בעלת מבנה מוגדר. הפרמטרים הבסיסיים ביותר הם משקלים ו-bias. הפרמטרים בעצם מחזיקים את ה"ידע" של הרשת ביחס למשימה הדרושה, והם, יחד עם ארכיטקטורת הרשת, מהווים את לב המודל.

על מנת לקבל פרמטרים טובים, מבצעים "אימון" של הרשת.

הקלטים/ דוגמאות עליהן המודל מתאמן מכונים Training Data Set (סט האימון). חלק מדוגמאות אלו מוגדרות כ-Validation Data Set, והמודל משתמש בהן במהלך האימון כמבחן עבור המודל, על מנת לכוונן את הפרמטרים. ישנו סט נוסף- Test Data Set, עליו המודל לא מתאמן, ומטרתו לבחון את איכות המודל וביצועיו ביחס לדוגמאות חדשות.

האימון מתבצע באופן בסיסי בשיטת Gradient Descent. זוהי שיטה איטרטיבית המשתמשת בגרדיאנט על מנת לעדכן את פרמטרי הרשת לביצוע אופטימיזציה שלהם. בסוף כל איטרציה מחושב loss המהווה מדד לטיב הפרמטרים. מתבצעת בדיקה של השפעת כל פרמטר ברשת על פונקציית ה-loss. בהתאם להשפעת כל פרמטר, הפרמטר מעודכן.

בכל איטרציה מחשבים את פלט הרשת בהינתן קלט, בודקים את איכות התוצאה (ע"י השוואה בין הפלט המתקבל לפלט הרצוי), משנים מעט את פרמטרי הרשת בהתאם לתוצאה זו (אופטימיזציה), וחוזר חלילה עד להגעה לתוצאות טובות (איור 4).

באופן מתמטי:

בתחילת האימון, הפרמטרים מאותחלים באופן רנדומלי.

בכל איטרציה:

1. מעבירים את הקלט $\bar{x}$ דרך הרשת לקבלת פלט y'.
2. מחשבים את השגיאה בין הפלט y' לפלט הרצוי y בעזרת פונקציית Loss. לדוגמא:

$$L(\bar{w}) = (y'(\bar{w}) - y)^2$$

3. מעדכנים כל משקל w_i:

$$w_i \leftarrow w_i + \Delta w_i$$

כאשר:

$$\Delta w_i = -\eta \frac{\partial L}{\partial w_i}$$

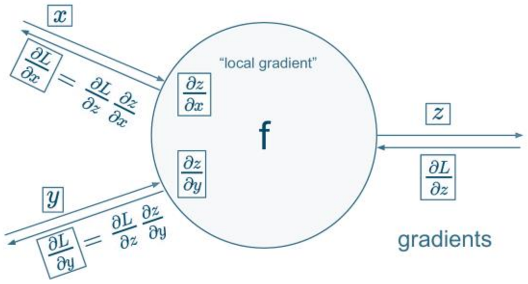
ניתן לראות כי בעצם מדובר בגרדיאנט:

$$\Delta w_i = -\eta \vec{\nabla} L(\bar{w})$$

בשיטה זו, המשקלים מעודכנים במטרה למזער את השגיאה (ה-Loss), משום שהביטוי $\frac{\partial L}{\partial w_i}-$ הוא בכיוון נקודת המינימום של ה-Loss (הגרדיאנט עצמו מצביע על כיוון המקסימום, ולכן יחד עם סימן שלילי הביטוי מצביע על הכיוון המנוגד).

η הוא ה-Learning Rate. זהו משתנה המציין את גודל הצעד בכיוון הנגדי לגרדיאנט בעת שינוי הפרמטרים.

את הנגזרת החלקית $\frac{\partial L}{\partial w_i}$ מחשבים בעזרת אלגוריתם Back-Propagation. אלגוריתם זה עושה שימוש בכלל השרשרת על מנת לחשב את נגזרת ה-Loss ביחס לכל פרמטר ברשת (איור 3).



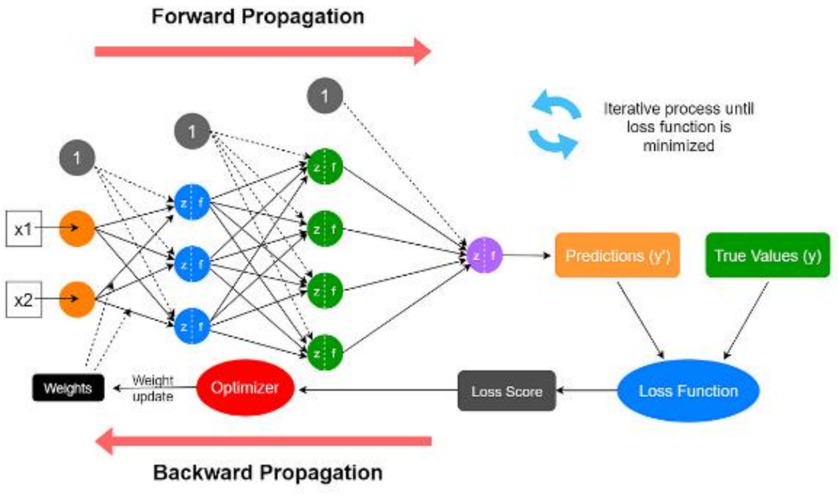
איור 3: כלל השרשרת. הכלל משמש עבור תהליך backpropagation.

ישנן מספר טכניקות אופטימיזציה (Optimization) מורכבות יותר ביחס ל-SGD:

Mini-Batch Gradient Descent, Momentum, Nesterov Accelerated Gradient (NAG), AdaGrad (Adaptive Gradient Algorithm), RMSprop (Root Mean Square Propagation), Adam (Adaptive Moment Estimation), L-BFGS.

הטכניקות הפופולריות ביותר בשל ביצועיהן הן Adam, RMSprop.

ישנן גרסאות רבות לטכניקות הנפוצות: AdamW, Nadam, AdaMax Radam ועוד.



איור 4: אימון רשת נוירונים. איטרציה באימון כוללת מעבר ברשת, חישוב פלט, חישוב loss, ביצוע backpropagation ועדכון הפרמטרים.

CNN (Convolutional Neural Network)- רשת קונבולוציה

רשת קונבולוציה (איור 6) היא סוג של רשת נוירונים, אשר בשכבות מסוימות שלה, המעבר בין שכבות הוא באמצעות קונבולוציה במקום באמצעות הכפלה במטריצה.

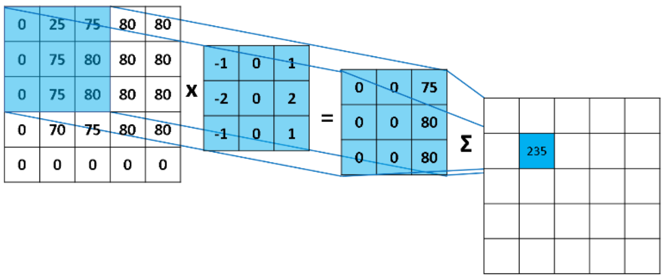
פעולת הקונבולוציה:

עבור הפעולה דרוש פילטר/ קרנל (kernel). מדובר במטריצת משקלים קטנה (3x3 , 5x5) אשר עומקה הוא כעומק הקלט (של שכבה זו). ניתן להשתמש במספר פילטרים בשכבת קונבולוציה. כל פילטר מניב שכבה בטנזור הפלט של שכבת הקונבולוציה. הפילטרים שונים ביניהם, אך כל אחד מהם יחיד עבור השכבה הוא מייצר.

הפילטר "מחליק" על גבי הקלט, ומבצע מכפלה סקלרית (Convolving) עם אזורים שונים בקלט (ניתן להוסיף bias לתוצאה). תוצאת כל מכפלה סקלרית היא סקלר בודד (איור 5). סקלר זה מהווה תא במטריצת הפלט. כל פילטר בשכבה מייצר מטריצת פלט, ומטריצות אלה משורשרות אחת אחרי השניה לקבלת טנזור פלט. מטריצות הפלט נקראות גם " activation maps". באופן דומה לשכבת fully connected, על פלט הקונבולוציה מפעילים פונקציית אקטיבציה.

פעולת הקונבולוציה באופן מתמטי:

$$\underbrace{O[i,j]}_{output} = \sum_{m}\sum_{n} \underbrace{I[i+m,j+n]}_{input} \cdot \underbrace{K[m,n]}_{kernel} + \underbrace{b}_{bias}$$



איור 5: פעולת הקונבולוציה. מתבצעת מכפלה סקלרית בין הפילטר לאזור בקלט.

גודל הפילטר, כמות הפילטרים וגודל הצעד (stride) בין האזורים המוכפלים בפילטר, ניתנים לבחירה. בנוסף, ניתן לבצע Padding בשלב הקונבולוציה: הוספת שורות ועמודות (לרוב אפסים) סביב קצות הקלט, המביאים לכך שיתבצעו יותר מכפלות, ומימדי הפלט יגדלו.

לאחר שכבה זו מקובל להוסיף שכבת Pooling. זוהי שכבה שמבצעת "הורדת קצב" (מקטינה את המימדים) של כל אזור קטן (לרוב 3x3 , 2x2) . שיטות Pooling נפוצות הן Max Pooling, Average Pooling.

עד כה מודלים תוארו כבאלו הפועלים בכל איטרציה על קלט בודד. בפועל, בכל איטרציה המודל מקבל כקלט קבוצת דוגמאות (Batch). המודל מופעל על כולן ומתקבלות תוצאות עבור כולן. הפרמטרים מעודכנים ביחס לכל התוצאות.

חשוב שאתחול פרמטרי המודל לא יבוצע בצורה רנדומית לחלוטין, משום שזה עלול להוביל לביצועים פחות טובים. קיימים מספר אתחולים מקובלים, ביניהם:

Xavier Initialization, He Initialization, Zero Initialization.

לרוב המודל מתאמן על מספר רב של דוגמאות. אולם אעפ"י שהוא מתאמן על מגוון דוגמאות, ישנה אפשרות שביצועיו יהיו טובים עבור הדוגמאות עליהן התאמן, ופחות טובים עבור דוגמאות עליהן לא התאמן (Overfitting). ישנן מספר פעולות Generalization שניתן לבצע על מנת להתמודד עם בעיה זו:

Dropout (כיבוי נוירונים מסוימים ברשת).

Batch normalization (נרמול המידע לפני כל שכבה).

Weight Decay (הקטנת ערכי פרמטרים/ משקלים ברשת).

Regularization (שינוי פונקציית ה-Loss כך שתגדל עבור משקלים גדולים).

Data Augmentation (הגדלת סט האימון ע"י ביצוע פעולות על סט האימון הקיים: סיבוב, שיקוף, מתיחה, כיווץ).

Cross-Validation (שינוי סט הוולידציה בכל מספר איטרציות).

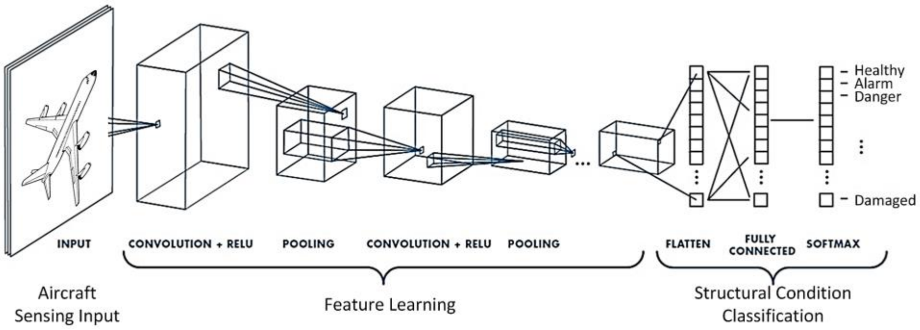
Early Stopping (עצירת האימון כאשר ביצועי המודל מגיעים לרוויה).

Ensembles (שילוב מספר מודלים).

ניתן להסביר את משמעות מבנה הרשת ומרכיביה כך:

מבחינת היררכיית שכבות, הפילטרים בשכבות הראשונות ברשת מזהים פיצ'רים בסיסיים. עבור תמונה למשל, מדובר בקווים, פינות, טקסטורות וכדומה. פילטרים בשכבות ביניים מזהים פיצ'רים מורכבים יותר- צורות וחלקי אובייקטים (עיניים, אף, זנב, גלגלים, חלונות, אותיות, מספרים...). ופילטרים בשכבות עמוקות מזהים פיצ'רים ברמה גבוהה, המחזיקים מידע לגבי הקלט כולו.

תפקיד המשקלים בכל שכבה ברשת הוא לקבוע את רמת החשיבות וההשפעה של המידע שמתקבל בקלט השכבה- עד כמה הוא חשוב ורלוונטי עבור השכבה הבאה ועבור פתרון הבעיה.



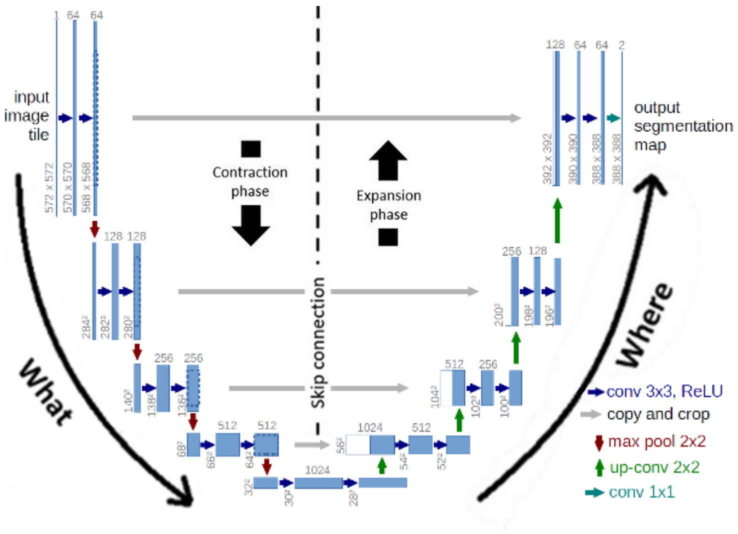
איור 6: ארכיטקטורת רשת קונבולוציה. הרשת מכילה שכבות קונבולוציה, pooling, ורשת fully connected.

ישנן רשתות קיימות בעלות ארכיטקטורה מורכבת בקפידה, אשר נבחנו ושופרו עד לביצועים טובים מאוד. רשתות נפוצות הן:

AlexNet, VGGNet, GoogLeNet, ResNet, DenseNet, MobileNet, NAS.

U-net

U net היא רשת CNN בעלת ארכיטקטורה שצורתה U: היא מכילה שני חלקים- מקודד (Encoder) ומפענח (Decoder). המקודד והמפענח מכילים כל אחד מספר שכבות קונבולוציה ו-pooling (איור 7). לרוב פונקציית האקטיבציה בארכיטקטורה זו היא ReLu. המקודד מבצע הורדת קצב (Contracting Path), ואילו המפענח מבצע העלאת קצב (Expanding Path). שני החלקים סימטריים ביניהם, ומחברים ביניהם skip connections אשר מעבירים מידע מהמקודד למפענח.[3]



איור 7: U-net. רשת הקונבולוציה בנויה בתצורת מקודד ומפענח כאשר בנוסף ישנם skip connections.

מימדי פלט הרשת הם לרוב כמימדי הקלט, כך שתמונה מומרת לתמונה אחרת. תכונה זו שונה מרשת CNN סטנדרטית, בה קלט הרשת הוא תמונה, אך הפלט הוא משפט (תיאור התמונה)/ סיווג (מה בתמונה)/ מילה (האם אובייקט קיים/ לא קיים).

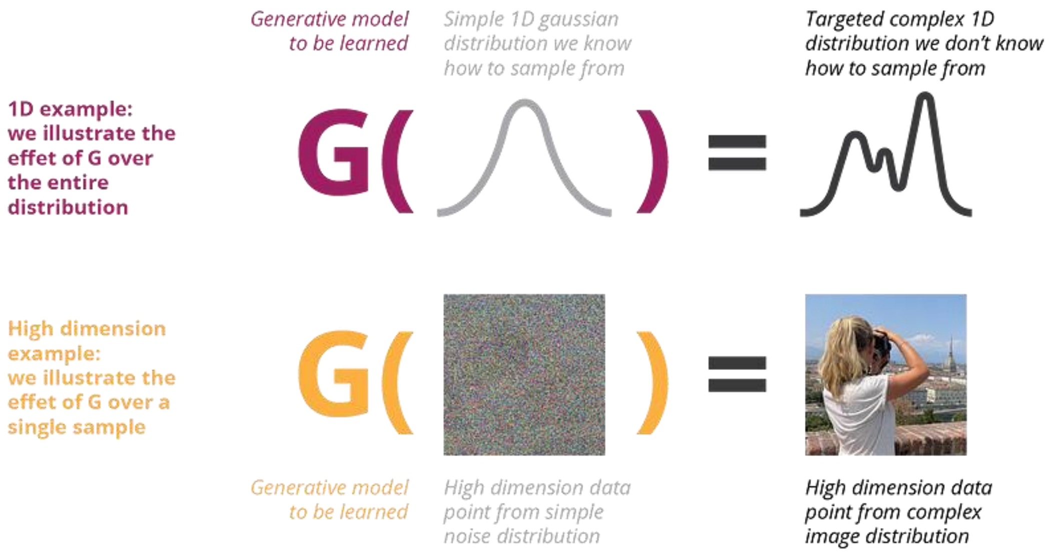
ניתן לשים לב כי תכונת המימדים הזהים נובעת בין היתר מכך שבשונה מרשת קונבולוציה רגילה, כאן אין שכבות (FC) Fully-Connected בסוף המבנה.

המקודד והמפענח הם מרכיבי הליבה של הרשת. תפקיד המקודד הוא ללמוד ולחלץ את הפיצ'רים החשובים בקלט, ותפקיד המפענח הוא לבצע גנרציה המבוססת על פיצ'רים אלו.

[3] Aladdin Persson, U-NET Paper Walkthrough, https://www.youtube.com/watch?v=oLvmLJkmXuc&t=363s

Diffusion model- מודל דיפוזיה

מודל דיפוזיה הוא מודל גנרטיבי ללמידת מכונה, המבוסס על מספר שלבים של גנרציה ועדכון בזה אחר זה. הוא עושה שימוש בתהליך סטוכסטי בדומה לשרשרת מרקוב. במהלך האימון המודל לומד את התפלגות סט האימון, ובהמשך הוא יכול ליצור דגימות חדשות מתוך התפלגות זו. הרעיון הבסיסי הוא להמיר התפלגות פשוטה הניתנת לדגימה בקלות (לרוב התפלגות גאוסית) להתפלגות מורכבת יותר (איור 8), תוך סדרת צעדים הפיכים.



איור 8: עיקרון מודל גנרטיבי. מטרת המודל ללמוד לעבור מהתפלגות פשוטה להתפלגות מורכבת.

ברגע שהמודל לומד את תהליך הטרנספורמציה, הוא יכול לייצר דגימות חדשות מתוך ההתפלגות המורכבת תוך התחלה מנקודה בהתפלגות הפשוטה, ו"פעפוע" (דיפוזיה) הדרגתי לעבר ההתפלגות המורכבת. הפעפוע כולל החסרת רעש הדרגתית מהדגימה הראשונית.

קיימות 2 גישות עליהן מתבסס מודל דיפוזיה:[4]

1. Denoising Diffusion Probabilistic Models (DDPMs).
2. Score-Based Generative Models (SGMs).[5]

שתי הגישות מורכבות משני תהליכים מרכזיים: forward process, backward process.

בשתי הגישות תהליך ה-forward משמש ללמידה, והוא מכיל מספר שלבים של הרעשת הדאטה עד להגעה לרעש גאוסי. תהליך ה-backward מכיל את אותה כמות שלבים, כאשר בכל שלב מתבצעת הפחתת רעש קטנה. כך מגיעים לגנרציה של דאטה חדש (מתוך רעש). מטרת המודל היא ללמוד כיצד להפחית רעש בכל שלב.

בגישת DDPMs המעבר בין השלבים מבוסס על תיאוריית שרשרת מרקוב.

בגישת Score-based, המעבר בין השלבים מבוסס על תיאוריית Langevin Dynamics. ישנה נוסחה איטרטיבית למעבר בין השלבים, והיא כוללת שימוש בפונקציית score.

[4]Encord Blog An Introduction to Diffusion Models for Machine Learning
https://encord.com/blog/diffusion-models /
[5]Song, Y., Durkan, C., Murray, I., & Ermon, S. (2021). Maximum likelihood training of score-based diffusion models. Advances in neural information processing systems, 34, 1415-1428.
https://papers.nips.cc/paper/2021/file/0a9fdbb17feb6ccb7ec405cfb85222c4-Paper.pdf
https://www.youtube.com/watch?v=wMmqCMwuM2Q

מקורות מוזיקה

בתחום האודיו, ניתן להתייחס לדגימת שמע כאל סכום של מקורות נפרדים:

$$\underbrace{\mathbf{y}} = \sum_{n=1}^{N} \underbrace{\mathbf{x}_n}$$

שלא כמו בתת-תחומים אחרים בתחום האודיו (למשל דיבור), מקורות מוזיקה (המרכיבים דגימת שמע) הם בעלי הקשר (context) ביניהם. הם חולקים מקצבים ומשתלבים זה עם זה בהרמוניה. מבחינה מתמטית תכונה זו מתבטאת בכך שההתפלגות המשותפת של המקורות $p(\mathbf{x}_1, \dots, \mathbf{x}_N)$ אינה שקולה למכפלת התפלגויות של המקורות הנפרדים $\{p_n(\mathbf{x}_n)\}_{n=1,\dots,N}$.

ידיעת ההתפלגות המשותפת $p(\mathbf{x}_1, \dots, \mathbf{x}_N)$ מרמזת ידיעה לגבי התפלגות ה-mix $p(\mathbf{y})$, משום שניתן להגיע אליה מתוך הסכום.

גנרציה של מוזיקה דורשת יכולת לסנתז כלים בנפרד ולשלבם. ניתן לשים לב כי הן בגנרציה והן בהפרדה, קיים מרכיב ההקשר בין הכלים. לכן הגיוני כי ישנו קשר הדוק בין שתי התחומים: גנרציה והפרדה.

חלופות לפתרון

ניתן להתייחס להפרדת מקורות כאל מקרה פרטי של גנרציה מותנית, כאשר תהליך הגנרציה מותנה ב-mix הנתון $\mathbf{y}(0)$. גנרציה זו דורשת את חישוב פונקציית ה-score של ההתפלגות הפוסטריורית:

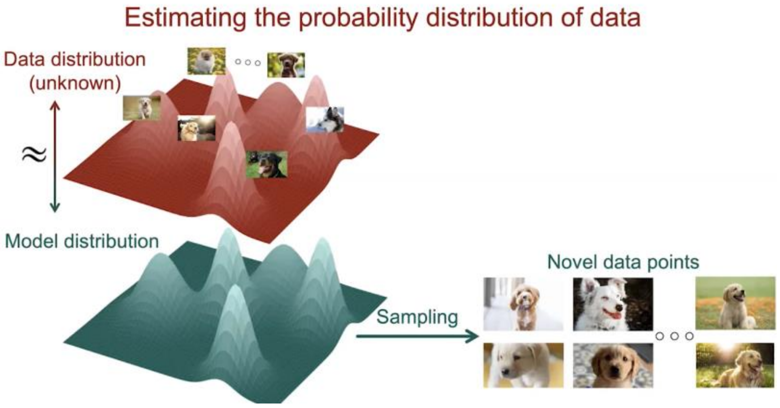
$$\nabla_{\mathrm{x}(t)} \log\, p(\mathbf{x}(t) |\, \mathbf{y}(0))$$

שיטות סטנדרטיות למימוש גנרציה מותנית עבור מודלי דיפוזיה כוללות שערוך ישיר של ה-score האפוסטריורית, או שערוך פונקציית הנראות (likelihood) $p\big(\mathbf{y}(0)\big|\mathbf{x}(t)\big)$ ושימוש בנוסחת בייס למציאת הפוסטריור. השיטה השניה כוללת לרוב אימון של מודל נפרד (בד"כ מסווג) עבור ה-score של פונקציית הנראות.

שיטת הפתרון

השיטה הנתונה במאמר מתייחסת לרשת המבוססת על מודל דיפוזיה גנרטיבי, שבאפשרותה לבצע את 3 המטרות כולן: גנרציה, גנרציה חלקית, והפרדה. אנו התייחסנו רק להפרדה, אולם השיטה משותפת לכל המטרות.

אנו השתמשנו במודל דיפוזיה בגישת score-based. כמודל דיפוזיה, מטרתו ללמוד תכונות של התפלגות של דאטה מסוים. המשימות שהמודל מנסה להשיג דורשות התפלגויות שאינן ידועות לנו, ומכיוון שהן ממימד גבוה, קשה גם לקרב אותן. עבור כל משימה נרצה לקרב תכונות של ההתפלגות האמיתית הנדרשת עבורה, שייצגו קירוב טוב של ההתפלגות כמתואר באיור 9.



איור 9: שערוך תכונות התפלגות ע"י מודל דיפוזיה. נרצה כי שערוך ההתפלגות (בכחול) יהיה כמה שיותר קרוב להתפלגות האמיתית (באדום).

הפתרון המעשי הוא לבצע score matching: לקרב את הגרדיאנט של לוגריתם ההתפלגות (score ההתפלגות), במקום את ההתפלגות עצמה. אולם הקירוב לא כולל את פונקציית ה-score כולה (על פני כל המרחב), אלא רק אזורים מסוימים שלה.

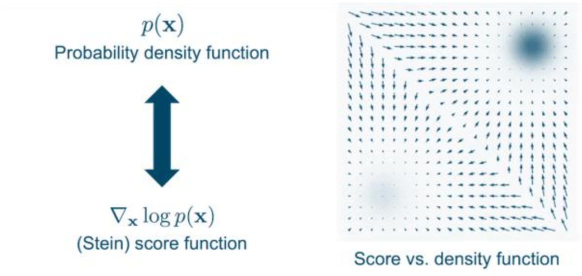
פונקציית ה-score ‏(probability density function)

עבור התפלגות $p(x)$ פונקציית ה-score מוגדרת כך:

$$score\ function: \quad \nabla_x \log p(x)$$

זהו שדה וקטורי המציין את הכיוונים בהם פונקציית הפילוג $p(x)$ גדלה. הפונקציה ידועה גם כגרדיאנט של ה-log-likelihood (איור 10).

ניתן לעבור בין פונקציית הפילוג $p(x)$ לפונקציית ה-score (ע"י גזירה/ אינטגרל).



איור 10: score function. הפונקציה היא גרדיאנט ה-log-likelihood של ההתפלגות.

מודל הדיפוזיה- Score-Based Generative Models

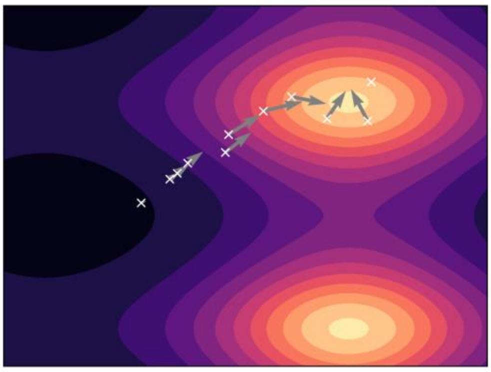
עבור התפלגות כללית $p(x)$, שימוש במודל לצורך גנרציה של דגימה מההתפלגות מתבצע באמצעות הנוסחה האיטרטיבית הבאה:

$$\mathbf{x}_{t-1} = \mathbf{x}_t + \alpha_t \cdot \underbrace{\nabla_\mathrm{x} \log p(\mathbf{x}_\mathrm{t})}_{score\ function} + \sigma_t \cdot \boldsymbol{\varepsilon}$$

כאשר הסיגנל הראשוני $\mathbf{x}_T$ הוא רעש גאוסי, והמטרה היא להגיע לסיגנל $\mathbf{x}_0$.

נוסחה זו מבוססת על תיאוריית Langevin dynamics. על פי תיאוריה זו, ניתן להגיע לדגימה מתוך התפלגות בעזרת הנוסחה האיטרטיבית. השיטה נקראת stochastic gradient Langevin dynamics (SGLD).

ניתן להתייחס לנוסחה האיטרטיבית כסוג של gradient ascent לעבר אזורים גבוהים התפלגות (איור 11). דגימה מתוך אזורים גבוהים של התפלגות היא דגימה אשר באופן מובהק נובעת מתוך ההתפלגות, ועל כך מתבססת התיאוריה.

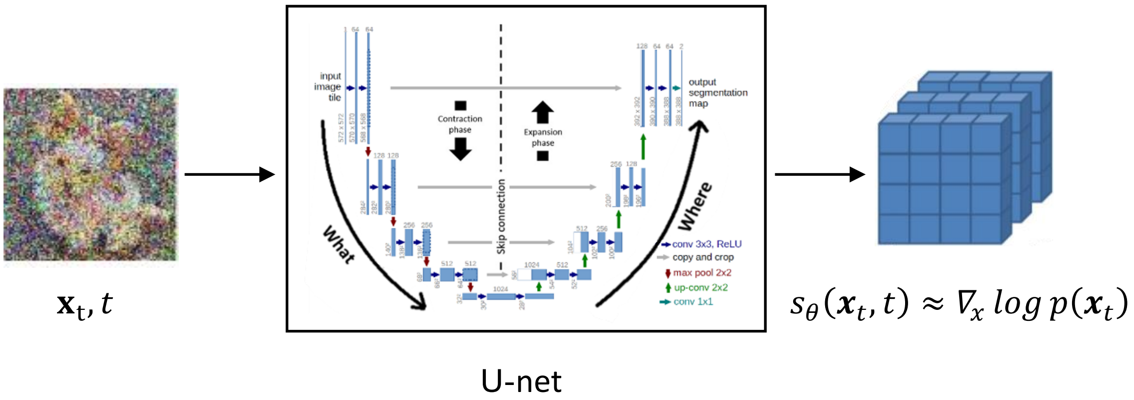


איור 11: SGLD. המסלול מוביל לעבר אזור גבוה בהתפלגות.

באמור, פונקציית ה-score $\nabla_{\mathrm{x}} \log p(\mathbf{x}_{\mathrm{t}})$ הדרושה עבור דגימה אינה נתונה לנו. אנו מאמנים רשת קונבולוציה בארכיטקטורת U[6], אשר מנסה לחזות את פונקציית ה-score של כל שלב. נתייחס להתפלגות כללית $p(x)$ (בהמשך נפרט לגבי ההתפלגויות הדרושות לכל משימה). פלט הרשת הוא $s_\theta(\boldsymbol{x}_t, t)$ (score-based model), פונקציה המשערכת את פונקציית ה-score (איור 12). כלומר נרצה לקבל:

$$s_\theta(\boldsymbol{x}_t, t) \approx \nabla_{\mathrm{x}} \log p(\boldsymbol{x}_t) \ \in \mathbb{R}^d$$

נשים לב כי הרשת מקבלת בקלט גם את מספר השלב t. הסיבה לכך היא שהרשת לא חוזה את פונקציית ה-score כולה, אלא רק את האזורים המתאימים לשלבים הכלולים בתהליך.

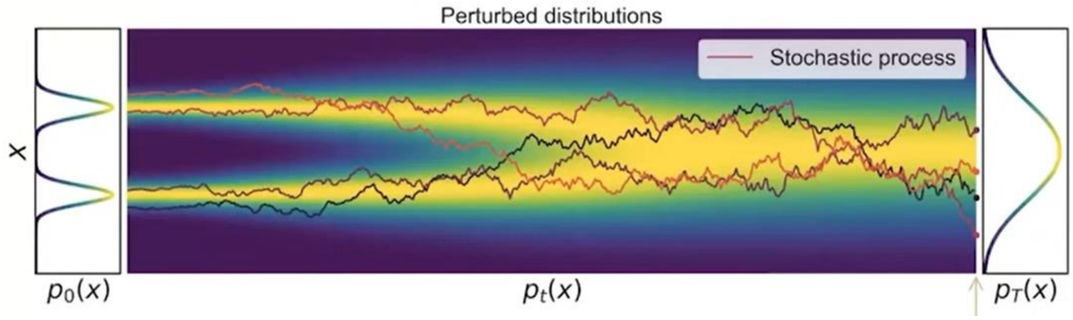


איור 12: שימוש ברשת הקונבולוציה. הרשת חוזה את פונקציית ה-score עבור כל שלב.

[6] https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

בתהליך ה-forward מוגדרים I שלבים (מסומן לעיתים כ-T) של הרעשת הקלט. בבל שלב ההתפלגות משתנה עקב הוספת הרעש. התפלגות השלב הראשון, המסומן כ-'0' היא $p(x)$. זוהי התפלגות הקלט הנקי. התפלגות השלב האחרון, המסומן כ-I, היא התפלגות גאוסית (רעש) סביב תוחלת אפס. בהתאמה, לכל שלב יש את פונקציית ה-score שלו.

תהליך ה-forward מתואר באיור 13, כאשר $I \to \infty$ (אינסוף שלבים- תהליך רציף).



איור 13: forward process. מתבצע מעבר בין 2 התפלגויות חד מימדיות, כאשר ההתפלגות הסופית היא התפלגות גאוסית.

תהליך ההרעשה הוא תהליך סטוכסטי[7], המכיל אינסוף מ"א $\{x_t\}_{t\in[0,T]}$ (t רציף). לכל משתנה יש את ההתפלגות שלו $\{p_t\}_{t\in[0,T]}$. המשוואה המתארת את התהליך היא SDE: Stochastic differential equation (משוואה דיפרנציאלית סטוכסטית). המשוואה:

$$dx = f(x,t)dt + g(t)dw$$

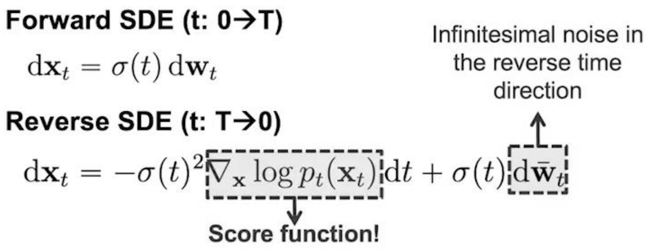
זו משוואה שדומה למד"ר (ODE), למעט איבר אקראי נוסף (w). ניתן להמיר את המשוואה למשוואה מקבילה פשוטה יותר:

$$dx_t = \sigma(t)dw_t$$

על מנת לבצע דגימה (גנרציה של דאטה חדש), נדרש למצוא את התהליך ההפוך, כלומר למצוא משוואה שהפוכה למשוואה זו.

[7] Stochastic process, in probability theory, a process involving the operation of chance - Written and fact-checked by Britannica.
Last Updated: Apr 12, 2024 • Article History.

לכל משוואת SDE, ישנה נוסחה עבור המשוואה ההופכית- "Reverse SDE":

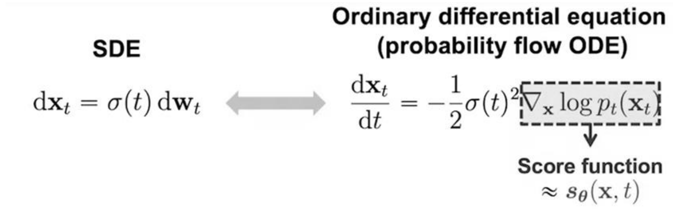


ניתן לפתור את ה-SDE עבור x:

$$x \leftarrow x - \sigma(t)^2 s_\theta(x,t)\Delta t + \sigma(t)' z \quad (z \sim N(0, |\Delta t| I))$$

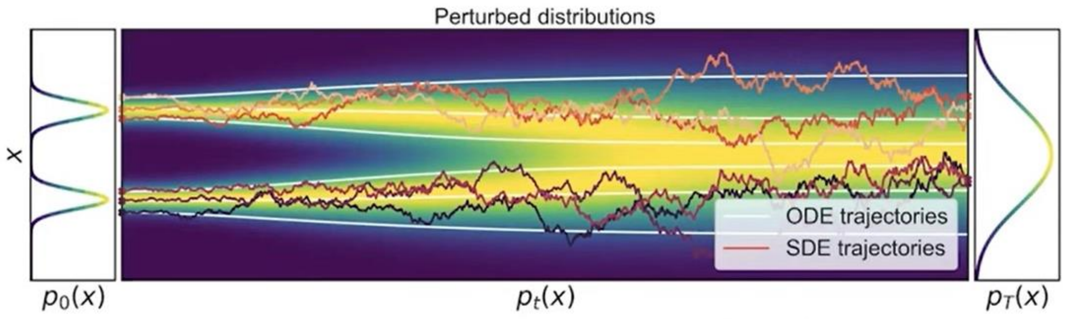
$$t \leftarrow t + \Delta t$$

ל-SDE כזו, יש מד"ר (ODE) מקבילה (איור 14) (המשמרת את מאפייני ה-SDE):



הערה: במאמר היא מתוארת כך:

$$dx(t) = -\sigma(t)\nabla_{x(t)} \log p\big(x(t)\big)\, dt$$



איור 14: ODE vs. SDE. ה-ODE משמרת את תכונות ה-SDE.

בכדי לבצע שערוך טוב, המודל משתמש ברעש מבוקר, אותו הוא מוסיף ומחסיר. ההתפלגות הינה:

$$\text{a gaussian perturbation kernel } : p(x_t|x_0) = N(x_t; x_0, \sigma_t^2 \boldsymbol{I})$$

כאשר $\sigma(t)$ מכוונן את כמות הרעש שנוספת לדאטה בכל שלב.

התהליך ההפוך (denoising) מתואר ע"י המד"ר:

$$dx(t) = \ \sigma(t)\nabla_{\mathrm{x(t)}} \log p\big(x(t)\big)\, dt$$

ניתן לפתור משוואה זו ולקבל נוסחה עבור x. הפתרון הבסיסי ביותר הוא:

- Sample from $p(\mathbf{x})$ using only the score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$
- Initialize $\mathbf{x}^0 \sim \pi(\mathbf{x})$
- Repeat for $t \leftarrow 1, 2, \cdots, T$

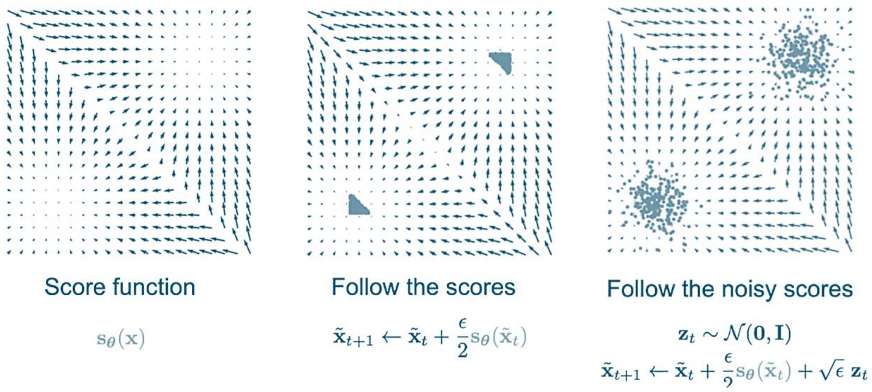
  $\mathbf{z}^t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

  $\mathbf{x}^t \leftarrow \mathbf{x}^{t-1} + \frac{\epsilon}{2}\nabla_{\mathbf{x}} \log p(\mathbf{x}^{t-1}) + \sqrt{\epsilon}\ \mathbf{z}^t$
- If $\epsilon \to 0$ and $T \to \infty$, we are guaranteed to have $\mathbf{x}^T \sim p(\mathbf{x})$

בפתרון זה, השימוש הוא ב-$s_\theta(x_t, \sigma_t)$ (המייצגת את רשת הנוירונים) במקום ב-$\nabla_{\mathrm{x}} \log p(\mathbf{x}_t)$. הרעיון הוא שאם תהליך ההרעשה (forward) מכיל I שלבים, ישנן I פונקציות פילוג (פונקציה לכל שלב), ובהתאמה ישנן I פונקציות score. עבור שלב i מקבלים מהרשת את $s_\theta(x, i)$ - פונקציית ה-score של אותו שלב. מקדמים את x R פעמים בכיוון פונקציה זו, בשאיפה להתכנסות פונקציית הפילוג של אותו שלב. מתקדמים ל-i הבא (שקטן יותר), ומבצעים שוב R קידומים. וחוזר חלילה, עד להגעה לשלב '0', כך שמתקבלת דגימה מפונקציית הפילוג $p(x)$.

ישנם 2 תיקונים שיש לבצע בתהליך שתואר לעיל:

1. אם כל הנקודות (הנקודות הן בסיס ההתפלגות במצב מסוים) נעות בכיוון הגרדיאנט, כולן יתכנסו לאותו מקום. לכן, לאחר ביצוע צעד בכיוון הגרדיאנט, יש להוסיף רעש אקראי לכל הנקודות (איור 15).



איור 15: תיקון תהליך העדכון. תנועה בכיוון הגרדיאנט בלבד (באמצע) תניב דגימות זהות. תוספת רעש (מימין) תניב דגימות שונות.

2. באזורים בהם צפיפות פונקציית ה-score נמוכה, קשה לחזות את הפונקציה. הפתרון לזה הוא הוספת רעש גאוסי. אך מתקבל כי רעש רנדומי לא מספיק, לכן משתמשים ברמות שונות של רעש $\{\sigma_i\}_{i\in\{0,\dots,I\}}$. בהתאמה לכך, יש לעדכן את הרשת - הרשת חוזה את פונקציית ה-score הרועשת, במקום את המקורית. לכן היא מקבלת כפרמטר את σ_i.

<u>גנרציה והפרדה</u>

עבור משימת הגנרציה המלאה (של מספר כלים, אשר סכימה שלהם תניב סיגנל הרמוני), ההתפלגות אותה יש לדגום היא ההתפלגות המשותפת של הכלים (prior) $p(\mathbf{x}_1, \dots, \mathbf{x}_N)$.

עבור משימת הפרדת כלים, ההתפלגות אותה יש לדגום היא ההתפלגות המשותפת של הכלים כאשר היא מותנית ב-mix $\mathbf{y}$ אותו רוצים להפריד (posterior): $p(\mathbf{x}_1, \dots, \mathbf{x}_N|\mathbf{y})$.
הפרדת הכלים מתקבלת ע"י ביצוע תהליך הגנרציה כאשר זו ההתפלגות אותה רוצים לדגום. המודל מבצע גנרציה של הכלים, כאשר גנרציית כל כלי מתחילה מסיגנל רעש. בזכות התניית ההתפלגות ב-$\mathbf{y}$, הכלים המיוצרים הם אותם כלים שאנו מעוניינים להפריד. בעצם ניתן להתייחס למשימת ההפרדה כאל משימת גנרציה מותנית.

המודל לומד במהלך האימון לחזות את פונקציית ה-score הדרושה למשימת הגנרציה. למידה זו מאפשרת לבצע את כל המשימות, משום שניתן לעבור בין ההתפלגויות תוך שימוש ביחס בייס:

$$\underbrace{p(\mathbf{x}|\mathbf{y})}_{posterior} \propto p(\mathbf{y}|\mathbf{x}) \cdot \underbrace{p(\mathbf{x})}_{posterior}$$

$$\Rightarrow \underbrace{\nabla_{\mathrm{x_m}} \log p(\mathbf{y}|\mathbf{x})}_{for\ source\ separation} \approx \nabla_{\mathrm{x}} \log p(\mathbf{y}|\mathbf{x}) + \underbrace{\nabla_{\mathrm{x}} \log p(\mathbf{x})}_{for\ generation}$$

לפי המאמר, ניתן לקבל את $\nabla_{\mathrm{x_m}} \log p(\mathbf{y}|\mathbf{x})$ המשמשת להפרדת כלים בעזרת הרשת באופן הבא:

$$\underbrace{\nabla_{\mathrm{x_m}} \log p(\mathbf{y}|\mathbf{x})}_{for\ source\ separation} \approx \nabla_{\mathrm{x}} \log p(\mathbf{y}|\mathbf{x}) + \underbrace{\nabla_{\mathrm{x}} \log p(\mathbf{x})}_{for\ generation} \approx \boldsymbol{S_m^\theta} - \boldsymbol{S_N^\theta}$$

<u>אופן פעולת המודל</u>

האימון:

בהינתן סט אימון המכיל דוגמאות של קבצי כלי נגינה, מייצרים קבצים נוספים בעזרת קבצים אלו. עבור T שלבי דיפוזיה, מייצרים דאטה תואם לכל שלב, ע"י הרעשה הדרגתית של סט האימון:

$$\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon_t$$

האימון מבוצע על שלב ה-forward, כלומר מעבר הדרגתי מדגימה לרעש גאוסי. בכל מעבר הרשת חוזה את פונקציית ה-score. השערוך מוצב בנוסחת פתרון המד"ר, ומתקבל הסיגנל הרועש יותר, המהווה את סיגנל השלב הבא.

באופן טבעי, על מנת לאמן את המודל כך שיוכל לחזות בצורה טובה את פונקציות ה-score של השלבים השונים, יש לקבוע פונקציית loss הכוללת את פונקציות ה-score האמיתיות ואת החיזוי שלהן. היינו מצפים למשל כי האימון יבוצע כך שמטרתו תהיה למזער פונקציה של Fisher divergence (מקושר לאינפורמציית פישר), כאשר Fisher divergence מוגדר כך:

$$\mathbb{E}_{p(\mathbf{x})}[||\nabla_{\mathbf{x}} \log p(\mathbf{x}) - s_\theta(\mathbf{x})||_2^2]$$

זהו L2 בין פונקציית ה-score האמיתית לבין ה-score based model. כלומר היינו מצפים לבצע score matching.

אולם, ישנה בעיה מרכזית: אין לנו גישה להתפלגות האמיתית $p(\mathbf{x})$, ומכאן נובע כי אין לנו גישה לביטוי $\nabla_{\mathbf{x}}\log\, p(\mathbf{x})$.

כדי להתגבר על בעיה זו, נעשה שימוש בשיטת denoising score-matching. בשיטה זו מקרבים את score הדגימות המורעשות ע"י הנוסחה:

$$\nabla_{\mathrm{x_t}} \log p(\mathbf{x}_t|\mathbf{x}_0) = -\frac{\mathbf{x}_t - \mathbf{x}_0}{\sigma_t^2}$$

ההשוואה מתבצעת בין ביטוי זה לבין פלט הרשת בכל שלב.

יש לציין כי על הרשת ללמוד לחזות את פונקציות ה-score של כל שלבי הדיפוזיה. בכל שלב באימון הרשת לומדת לחזות את הפונקציות של כל השלבים, ופונקציית ה-loss כוללת את כלל השגיאות עבור כל השלבים.

בשלבים שונים מתבצעת הפחתת רעש בכמות שונה. בשלבים הקרובים להתפלגות הגאוסית ישנה הפחתת רעש משמעותית, ובשלבים הקרובים לשלב הסופי (בתהליך ה-backward) מתבצעים תיקוני רעש עדינים. הרשת לומדת את הקשר בין מספר השלב t לבין כמות הרעש הדרושה להפחתה באותו שלב.

השימוש במודל:

באופן כללי בשימוש במודל אנו מכניסים בקלט קובץ אודיו, ומקבלים קובץ אודיו אחר. הקובץ המתקבל יכול להיות דגימה חדשה מתוך התפלגות כלי מסוים (גנרציה), או, כמו במקרה שלנו, כלים מופרדים שמקורם בקובץ המהווה mix ביניהם.

הקובץ החדש מתקבל על ידי שימוש במודל הדיפוזיה, המכיל מספר רב של שלבים (~ 100), כאשר בכל שלב מקבלים סיגנל שונה. בסוף התהליך מקבלים את הסיגנל הרצוי.

המעבר בין השלבים הוא במסגרת תהליך ה-backward של המודל (reverse diffusion). בכל שלב מתבצעת הפחתת רעש לקבלת סיגנל חדש.

כל שלב במעבר מכיל שני תתי שלבים:

- בשלב הראשון ה-U-Net מקבלת בקלט את הסיגנל הנוכחי ואת מספר השלב (t). הרשת מוציאה כפלט שערוך של 2 פונקציות ה-score הדרושות של אותו שלב- S_m^θ, S_N^θ. ישנה לולאה עבור m, כאשר כל m מסמן כלי אחר.
- בשלב השני מתבצע עדכון של הסיגנל משלב t, כך שמתקבל הסיגנל המשוערך המתאים לשלב הבא (t-1). העדכון מתבצע כך:

$$\mathbf{x}_{t-1} = \mathbf{x}_t + \alpha_t \cdot \underbrace{S_m^\theta - S_N^\theta}_{\approx \nabla_{\mathbf{x}_m} \log p(\mathbf{x}_t|\mathbf{y})} + \sigma_t \cdot \boldsymbol{\varepsilon}$$

העדכון מהווה החסרת רעש מסיגנל השלב הקודם. ניתן לראות בנוסחה כי כמות הרעש אותה יש להחסיר תלויה באופן ישיר בפונקציית ה-score של השלב. כלומר חיזוי פונקציית ה-score וקביעת כמות הרעש הדרושה להפחתה, מהוות בעצם משימה זהה.

:MSDM Dirac

בעת נתייחס למשוואות הספציפיות עבור המודל שלנו. ניתן לקרב את פונקציית ה-score ע"י מידול $p(\mathbf{y}_t|\mathbf{x}_t)$ (posterior) כפונקציית דלתא של דיראק שממרוכזת סביב $\sum_{n=1}^{N} \mathbf{x}_n(t)$:

$$p(\mathbf{y}(t)|\mathbf{x}(t)) = \mathbb{1}_{\mathbf{y}(t) = \sum_{n=1}^{N} \mathbf{x}_n(t)}$$

השיטה נקראת "MSDM Dirac", ומנסים לשערך בה את:

$$\nabla_{\mathbf{x}_{m(t)}} \log p(\mathbf{x}(t) \mid \mathbf{y}(0)) \approx S_m^\theta((\mathbf{x}_1(t), \ldots, \mathbf{x}_{N-1}(t), \mathbf{y}(0) - \sum_{n=1}^{N-1} \mathbf{x}_n(t)), \sigma(t))$$

$$-S_N^\theta((\mathbf{x}_1(t), \ldots, \mathbf{x}_{N-1}(t), \mathbf{y}(0) - \sum_{n=1}^{N-1} \mathbf{x}_n(t)), \sigma(t))$$

באשר:

$\mathbf{x}_n(t) = \mathbf{y}(0) - \sum_{n=1}^{N} \mathbf{x}_n(t)$, $1 \le b \le N-1$, והביטויים S_N^θ, S_m^θ מציינים את הערכים של רשת ה-score לפי מקורות מספר m, N.

אלגוריתם

הפרדת המקורות מבוצעת באמצעות אלגוריתם sampler "MSDM Dirac" (איור 16).

Algorithm 1 'MSDM Dirac' sampler for source separation.

Require: I number of discretization steps for the ODE, R number of corrector steps, $\{\sigma_i\}_{i\in\{0,\dots,I\}}$ noise schedule, S_{churn}

```
Initialize x̂ ~ N(0, σ_I^2 I)
α ← min(S_churn / I, √2 − 1)
for i ← I to 1 do
  for r ← R to 0 do
    σ̂ ← σ_i · (α + 1)
    ε ~ N(0, I)
    x̂ ← x̂ + sqrt(σ̂^2 − σ_i^2) ε
    z ← [x̂_{1:N−1}, y − Σ_{n=1}^{N−1} x̂_n]
    for n ← 1 to N − 1 do
      g_n ← S_n^θ(z, σ̂) − S_N^θ(z, σ̂)
    end for
    g ← [g_1, ..., g_{N−1}]
    x̂_{1:N−1} ← x̂_{1:N−1} + (σ_{i−1} − σ̂) g
    x̂ ← [x̂_{1:N−1}, y − Σ_{n=1}^{N−1} x̂_n]
    if r > 0 then
      ε ~ N(0, I)
      x̂ ← x̂ + sqrt(σ_i^2 − σ_{i−1}^2) ε
    end if
  end for
end for
return x̂
```

איור 16: אלגוריתם MSDM Dirac.

ניתן לזהות את הנוסחה האיטרטיבית:

$$\boldsymbol{x}_{t-1} = \boldsymbol{x}_t + \alpha_t \cdot \underbrace{S_m^\theta - S_N^\theta}_{\approx \nabla_{\mathbf{x}_m} \log p(\mathbf{x}|\mathbf{y})} + \sigma_t \cdot \boldsymbol{\varepsilon}$$

שלבי האלגוריתם:

1. מוגדרים I צעדים עבור משוואת המד"ר, R צעדים עבור אופטימיזציית האלגוריתם, σ_i מוגדר כרעש מאותחל ונתון בצעד i, S הרשת עצמה.
2. אתחול $\hat{x}$ (מקור גנרטיבי) בהתפלגות נורמלית של הרעשים הרנדומלים.
3. α מתעדכן למינימום בין $\sqrt{2}-1$, S_{churn}[8].

[8] Constrained Source – the instruments. It is a mechanism used in the Euler ODE discretization logic.

4. לולאה העוברת על הצעדים במד"ר.
5. לולאת אופטימיזציה.
6. מעדכנים את $\hat{\sigma}$ להיות $\sigma_i(\alpha+1)$.
7. הגדרת ϵ כהתפלגות רעש נורמלית.
8. עדכון $\hat{x}$ על ידי הוספת שגיאה הממושקלת לפי $\hat{\sigma}, \sigma_i$.
9. עדכון וקטור $\boldsymbol{z}$ כמערך המכיל את $\hat{x}$ ללא המקור הגנרטיבי האחרון, ושרשור מקור אחרון הנובע מהפרש בין המקורות המקוריים לגנרטיביים.
10. לולאה על n.
11. נוצר ה - gradient (מתעדכן במשתנה g_n), על ידי הפרש בין $S_n^\theta(\boldsymbol{z},\sigma) - S_N^\theta(\boldsymbol{z},\sigma)$ כאשר S^θ היא ערך פונקציית הscore. ההפרש מייצר את ה-score based, שהיה בין מקור מסוים למקור האחרון (הגנרטיביים), כלומר הפרשת פונקציית ה-score.
12. סוף לולאה.
13. שרשור g_n לווקטור $\boldsymbol{g}$.
14. Gradient decent – הצעד מתאפיין על ידי שגיאת השונויות.
15. עדכון $\hat{x}$.
16. לולאה עבור שלבי אופטימיזציה (לפי R).
17. מגדירים ϵ כהתפלגות נורמלית סביב 0 עם שונות של 1.
18. מעדכנים את $\hat{x}$ ע"י הוספת המרחק בין הרעש של צעד הקודם לצעד הנוכחי.
19. סוף לולאה.
20. סוף לולאה.
21. סוף לולאה.
22. החזרת $\hat{x}$.

הניסוי

<u>dataset</u>

סט האימון עבור המודל הינו Slakh2100 (Synthesized Lakh), מערך נתונים סטנדרטי להפרדת מקורות מוזיקה.

הסט כולל 2100 טרקים (רצועות מוזיקה) כאשר החלוקה הינה:

- 1500 טרקים עבור אימון.
- 375 עבור ולידציה.
- 225 עבור טסט.

הסט המלא מכיל כ-30 כלים. אצלנו נעשה שימוש בארבעה כלים נפוצים במוזיקה: בס, תופים, גיטרה ופסנתר. מכל 30 הכלים, לא כולם נוכחים לאורך כל השיר. נוכחות 4 הכלים הנבחרים היא גבוהה: בס- 94.7% , תופים- 99.3% , גיטרה- 100% , פסנתר- 99.3%.

הסט מכיל כמות גדולה מאוד של נתונים ביחס למערכי נתונים אחרים (למשל MusDB). כמות הנתונים היא מרכיב מכריע עבור איכות המודל הגנרטיבי המתקבל, מה שהופך את הסט לבחירה טובה.

<u>ארכיטקטורה</u>

ארכיטקטורת הרשת הינה גרסה של "Moûsai". ישנה U-net המשולבת עם רשתות CNN חד מימדיות בעלות self-attention. ה-U-net מכילה מקודד (encoder), צוואר בקבוק (bottleneck), מפענח (decoder), ו-skip connections. המקודד מכיל 6 שכבות המורכבות מ-2 בלוקי קונבולוציה מסוג ResNet, ובנוסף attention מתצורת multi-head ב-3 השכבות האחרונות.

אמצעי החקר

אופן הערכת איכות הפרדת המקורות נעשה באופן מתמטי באמצעות מטריקת $\mathrm{SI-SDR_i}$ (scale-invariant Signal-to-Distortion Ratio improvement).[9] מטריקה זו היא שיפור של מדד $\mathrm{SI-SDR}$, אשר בוחן איכות סיגנל ביחס לסיגנל רצוי. המדד המשופר עמיד בפני שינויים באמפליטודה (scaling) של הסיגנלים.

עבור אות מקורי $\mathbf{x}_n$ ואות משוערך $\hat{\mathbf{x}}_n$, המדד $\mathrm{SI-SDR}$ מוגדר כך:

$$\mathrm{SI-SDR}(\mathbf{x}_n, \hat{\mathbf{x}}_n) = 10\log_{10}\frac{\| \alpha\mathbf{x}_n \|^2 + \epsilon}{\| \alpha\mathbf{x}_n - \hat{\mathbf{x}}_n \|^2 + \epsilon}$$

כאשר:

$$\alpha = \frac{\mathbf{x}_n^T\hat{\mathbf{x}}_n + \epsilon}{||\mathbf{x}_n||^2 + \epsilon}\ , \qquad \epsilon = 10^{-8}$$

עבור זה, המדד המשופר $\mathrm{SI-SDR_i}$ בו השתמשנו מוגדר כך:

$$\mathrm{SI-SDR_i} = [\mathrm{SI-SDR}(\mathbf{x}_n, \hat{\mathbf{x}}_n)] - [\ \mathrm{SI-SDR}(\mathbf{x}_n, \mathbf{y})]$$

הביצועים טובים כאשר המדד גבוה.

אופן השימוש

בשימוש ברשת (test) אנו מכניסים ארבעה קבצים נקיים של ארבעת הכלים, כל אחד באורך 5:11 דקות. מתבצע mix בין הכלים, והרשת מנסה להפריד ביניהם. פלט הרשת הוא 26 תיקיות, כאשר כל תיקיה מתייחסת למקטע בן 12 שניות מתוך האורך המלא (5:11). בכל תיקיה ישנם ארבעה קבצים (" piano ", " guitar", "drums", "bass"), המהווים את אותו מקטע שהופרד לארבעת הכלים. בנוסף מתקבלת מטריצה עם מדדי $\mathrm{SI-SDR_i}$ לכל כלי: מדד לכל 88200 דגימות (147 מדדים). לכל כלי ישנו גם ממוצע המדדים שלו לאורך הקטע המלא.

---

[9] Scale-invariant signal-to-distortion ratio improvement (SI-SDRi). https://oecd.ai/en/catalogue/metrics/scale-invariant-signal-to-distortion-ratio-improvement-si-sdri

<u>הממשק</u>

הרצת הקוד נעשתה בשרת מרוחק אליו התחברנו דרך טרמינל SSH "MobaXterm" תוך הפעלת חיבור מאובטח בעזרת VPN. הקוד הוא מתוך פרויקט GitHub בשם "multi-source-diffusion-models" שבוצע עבורו clone לתוך השרת.[10] הריצה בוצעה על שרתי GPU שונים של חברת NVIDIA.

<u>מהלך הניסוי</u>

ראשית הרצנו evaluation (test) כאשר ה-checkpoints (המשקלים) של הרשת הם אלו הנתונים בקוד הנלווה למאמר.

שנית ביצענו אימון של הרשת ללא שום שינוי ברשת עצמה, והרצנו evaluation לשם בקרה- האם תוצאות אלו תואמות לתוצאות המאמר.

ביצענו לאחר מכן אימון זהה נוסף לשם בקרה נוספת ובדקנו את התוצאות.

בשלב האחרון רצינו לשפר את התוצאות הנתונות, ולשם כך בדקנו מספר גישות לשיפור.

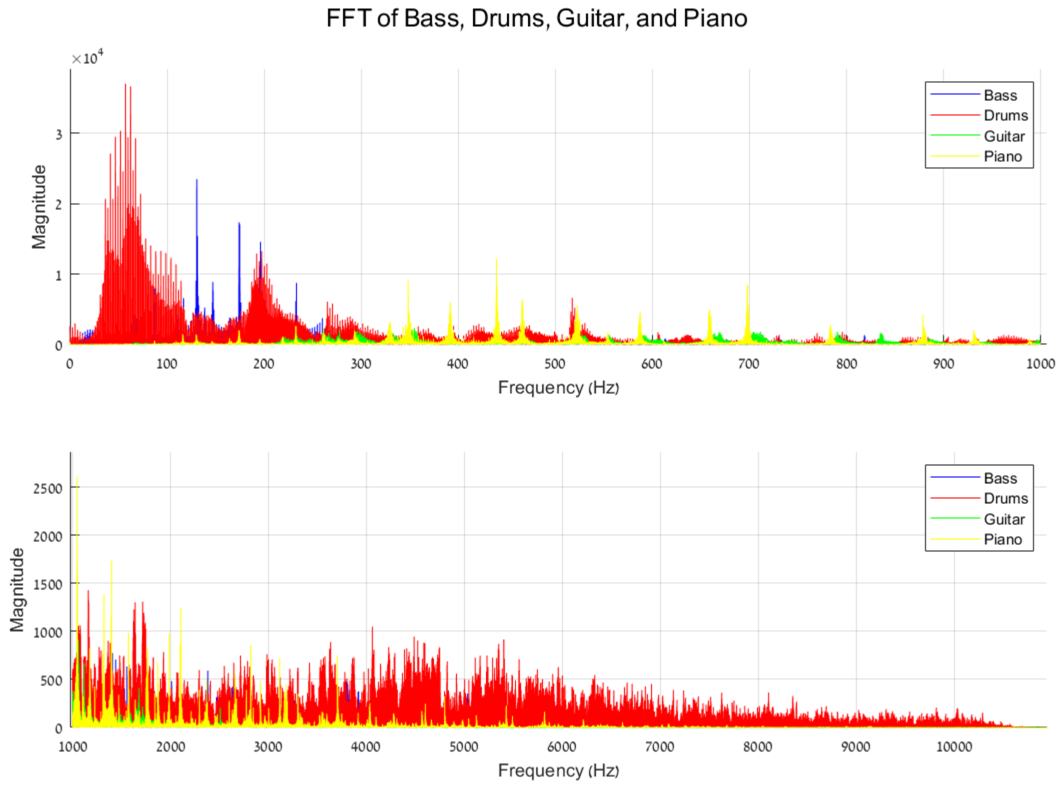
---

[10] https://github.com/gladia-research-group/multi-source-diffusion-models
https://github.com/archinetai/audio-diffusion-pytorch/tree/main

<u>שיטות שיפור התוצאות</u>

לאחר השימוש במודל וקבלת התוצאות (כלים מופרדים), רצינו לשפר את ההפרדה.

כפעולה מקדימה לבחירת שיטת שיפור ביצענו התמרת FFT על מנת לקבל מידע על אופי הדאטה (קלט הרשת) מבחינה תדרית (איור 17).



איור 17: התמרת FFT של הכלים. ישנה חפיפה רבה בין הרכבי הכלים השונים.

ההרכבים התדריים של הכלים השונים חופפים בחלקם הגדול זה לזה, כך שגישת סינון תדרים אינה מהווה פתרון טוב- גם לא על פלט הרשת. אמנם ההרכב התדרי של פלט הרשת מתאים מעט יותר לסינון, אולם לאחר ביצוע מעט ניסויים לצורך ניקוי רעשי הרקע של פלט הרשת, החלטנו שלא להמשיך בדרך זו.

גישה אפשרית לשיפור היא להעביר את פלט הרשת פעם נוספת ברשת. לא השתמשנו בשיטה זו.

שיטת השיפור בה בחרנו היא שינוי פונקציות ומרכיבים ברשת עצמה. ידוע כי ישנה חשיבות מכרעת למרכיבים כגון:

- פונקציית loss.
- סוג Optimizer.

השתמשנו באפשרויות הללו על מנת לנסות ולשפר את התוצאות.

שינוי מרכיבים ברשת

1. פונקציית loss:

פונקציית הפסד (loss function) היא פונקציה שמטרתה למדוד עד כמה התחזיות של המודל תואמות את הנתונים האמיתיים. היא נותנת אינדיקציה על הדיוק של המודל במהלך האימון ומנחה אותו לשפר את התחזיות. כאשר מודל למידת מכונה מבצע תחזיות, פונקציית ההפסד משווה את התחזיות האלה לערכים האמיתיים (ה- ground truth). פונקציית ההפסד מחשבת את השגיאה בין התחזיות לערכים האמיתיים, ומספקת מדד כמותי לכמה רחוקות התחזיות מהתוצאה הנכונה. מטרת האימון של המודל היא למזער את פונקציית ה-loss, כלומר לגרום לתחזיות להיות קרובות יותר לערכים האמיתיים.

פונקציית ה-loss שמשתמשים בה בקוד המאמר היא MSE – mean square error, שגיאה ריבועית ממוצעת. זוהי פונקציה שבדרך כלל משתמשים בה עבור משימות רגרסיה, בהן המודל מנבא ערכים רציפים. משוואתה היא עבור אות $\mathbf{y}$ ואות משוערך $\hat{\mathbf{y}}$:

$$\text{MSE} = \left(\frac{1}{n}\right) \sum_{i=0}^{n-1} (\mathbf{y_i} - \hat{\mathbf{y}}_\mathbf{i})^2$$

כאשר yi הוא הערך האמיתי ו-$\widehat{y_i}$ הוא הערך המשוערך, ו-N היא כמות הדאטה. תכונה בולטת של פונקציה זו היא שהיא נותנת את אותה החשיבות לכל הדוגמאות המשוערכות שבאימון, ולכן פחות רגישה לדוגמאות קיצוניות בעלות loss גבוה.

תכונה זו מהווה מחד את יתרונה, שהוא מניעת overfitting לסט האימון ויכולת להתאים את המשקלים שייווצרו לאחר האימון עבור מגוון דוגמאות רחב, וביחד עם זאת, התכונה הזו יכולה להוות חיסרון, שכן גם אם דוגמה מסוימת שוערכה עם ערך loss שגדול או קטן מאד ביחס

לערכי ה-loss של דוגמאות אחרות, הוא עדיין מקבל את אותה החשיבות ובבך הסיכוי לכך שהמשקלים יהוו התאמה טובה ביותר פוחת.

ישנה פונקציית loss מוכרת נוספת – Cross-Entropy. משתמשים בה בעיקר עבור בעיות סיווג בינארי וסיווג multi-class. משוואתה היא עבור אות **y** ואות משוערך **x**:

$$l(\mathbf{x},\mathbf{y}) = \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} \omega_{y_n} \cdot \mathbb{1}_{\{y_n \neq ignore_{index}\}}} \cdot l_n$$

$$l_n = -\,\omega_{y_n} \log \frac{\exp(\mathbf{x}_{n,y_n})}{\sum_{c=1}^{C} \exp(\mathbf{x}_{n,c})} \cdot \mathbb{1}_{\{y_n \neq ignore_{index}\}} \; , C = Num\ of\ Classe = 4$$

משוואה זו למעשה מורכבת מהפרש של שתי התפלגויות הסתברות – הערכים האמיתיים והערכים המשוערכים. היא מייצרת איבר loss גבוה יותר עבור שערוכים שגויים, על מנת למנוע משערוכים שגויים אלו להוות השפעה משמעותית על ערכי המשקלים המחושבים במהלך האימון. כאן הוא יתרונה על פני שיטת MSE, אך עם זאת בה עלולה להיווצר בעיית overfitting של המשקלים לסט האימון.

בעיית הפרדת הכלים בפרויקט שלנו אכן נוגעת בסוג הלמידה שיש להשתמש בו במשימת קלסיפיקציה, שכן ישנם ארבעה כלים שונים שיש לזהות בתוך היצירה בכל אחד מחלקיה, שאף במאמר מכונים "source class".

במאמר אחר[11] מבוצעת השוואה של מודלים שונים שאומנו עם MSE ועם Cross-Entropy (CCE), ומראה ש-CCE מביא לעתים קרובות להתכנסות מהירה יותר ודיוק (accuracy) גבוה יותר, במיוחד במשימות Multi-Class בעלות מספר classes גבוה. מוסבר שם, שתוצאה זו נובעת בעיקר מהיכולת של CCE לספק גראדיינטים חדים יותר ואינפורמטיביים יותר וכך גם מתכנסת יותר במהירות, בהשוואה ל-MSE, שנוטה לייצר שיפועים קטנים יותר, ולהאט את קצב האימון.

---

[11] Evaluation of Neural Architectures Trained with Square Loss vs Cross-Entropy in Classification Tasks" ,Like Hui California San Diego La Jolla

במאמר נוסף[12] מתואר מחקר על רשת נוירונים בעלת שכבה בודדת (single-layer), שבו מצוין כי במחקר קודם שנערך באוניברסיטת MIT נמצא שרשת מסוג זה המאומנת עם פונקציית הפסד MSE, הניבה תוצאות טובות במשימות סיווג בינארי, אך לא במשימות סיווג Multi-Class, ומכאן היה הרושם המוטעה על רשתות מסוג זה שאינן טובות בסיווג שאינו ליניארי. אולם, במאמר הנ"ל בוצע האימון על רשת כזו עם פונקציית ההפסד CCE עבור משימת הפרדה שאינה ליניארית, והדבר הניב תוצאות טובות והתאמה מדויקת בהרבה יותר. גם שם, ההשערות להעדפת CCE על פני MSE דומות – הטענה היא ש-CCE מתאים יותר למוצאים הסתברותיים ומספק שיפועים חדים יותר לאימון רשתות נוירונים, מה שמוביל להתכנסות מהירה ויעילה יותר מאשר בשימוש ב-MSE.

על סמך הנאמר לעיל, ובשל קווי הדמיון הרבים של המשימה אותה אנו חוקרות למשימות multi-classification והפרדה שאינה ליניארית, החלטנו לבדוק האם שימוש בפונקציית ההפסד מסוג CCE תניב התאמה טובה ומדויקת יותר למשימת הפרדת הכלים לעומת פונקציית ההפסד MSE שבה השתמשו במאמר המקורי.

2. אופטימיזציית ADAMW

בחלק האימון, נעשה שימוש באלגוריתמי אופטימיזציה על מנת להביא את משקלי הרשת לאופטימליים ביותר למטרת פתרון הבעיה.

במאמר נעשה שימוש בשיטת ADAM

שיטת Adam (Adaptive Moment Estimation) היא אלגוריתם אופטימיזציה הנמצא בשימוש נרחב בלמידה עמוקה, שבאמור מטרתו לשפר את יעילות ודיוק האימון של רשת הנוירונים. אלגוריתם זה הוצג על ידי Diederik Kingma וע"י Jimmy Ba ב-2014. הוא משלב את היתרונות של שני אלגוריתמים אחרים – AdaGrad ו-RMSProp, שהם הרחבות של שיטת gradient descent, אחת המשתמשת בגרדיאנט מסדר ראשון ואחת בגרדיאנט מסדר

---

[12] Kamaledin Ghiasi-Shirazi, Competitive Cross-Entropy Loss: A Study on Training Single-Layer Neural Networks for Solving Nonlinearly Separable Classification Problems,2018, Ferdowsi University of Mashhad (FUM), Office No.: BC-123, Azadi Sq., Mashhad, Khorasan Razavi, Iran,

שני, מה שהופך אותו למתאים במיוחד למאגרי נתונים גדולים ולמודלים עם מבנה פרמטרים מורכב.

מאפיינים מרכזיים של Adam:

- קצבי למידה (learning rates) אדפטיביים: Adam מתאים קצבי למידה לכל פרמטר בנפרד, על בסיס המומנטים הראשון והשני (שהם בהתאמה הממוצע והשונות) של הגרדיאנטים. תכונה זו מבטיחה שפרמטרים עם גרדיאנטים גדולים יקבלו עדכונים קטנים יותר ולהיפך, דבר שמייצב את תהליך האימון.

- מומנטום: Adam שומר על קצב דעיכה אקספוננציאלי של גרדיאנטים קודמים (ממוצע) וגרדיאנטים בריבוע (שונות), מה שמוסיף אלמנט של מומנטום, המאיץ את האימון. זה עוזר להפחית את התנודות ולזרז את ההתכנסות.

- תיקון bias: Adam כולל תיקונים להטיה (bias) בשלבים הראשוניים של האימון, כדי לאפשר אומדנים מדויקים יותר לגרדיאנטים.

כללי העדכון בשיטת ADAM הם לפי המשוואות הבאות :

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

כאשר:

gt הוא הגרדיאנט של פונקציית המטרה בשלב t.

Mt הוא המומנט הראשון (ממוצע) של הגרדיאנטים,

Vt הוא המומנט השני (שונות) של הגרדיאנטים,

Beta1, beta2 הם קצבי דעיכה (בדרך כלל 0.9 ו-0.999),

Etha הוא קצב הלמידה,

Epsilon הוא ערך קטן שנוסף ליציבות נומרית.

ADAMW הוא ורסיה של ADAM.[13]

[13] Llugsi, R., El Yacoubi, S., Fontaine, A., & Lupera, P. (2021, October). Comparison between Adam, AdaMax and Adam W optimizers to implement a Weather Forecast based on Neural Networks for the Andean city of Quito. In 2021 IEEE Fifth Ecuador Technical Chapters Meeting (ETCM) (pp. 1-6). IEEE.

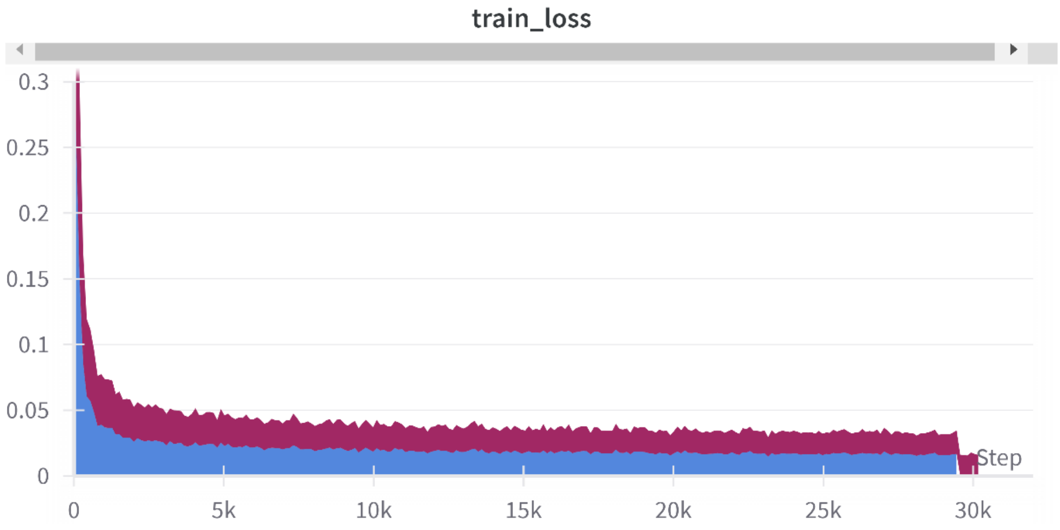
תוצאות

התוצאות (מדדי $\mathrm{SI} - \mathrm{SDR_i}$) כפי שהן מובאות במאמר:

Model	Bass	Drums	Guitar	Piano	All
Demucs (Défossez et al., 2019; Manilow et al., 2022)	15.77	19.44	15.30	13.92	16.11
Demucs + Gibbs (512 steps) (Manilow et al., 2022)	17.16	19.61	**17.82**	**16.32**	**17.73**
Dirac Likelihood					
ISDM	18.44	20.19	13.34	13.25	16.30
ISDM (correction)	**19.36**	**20.90**	14.70	14.13	17.27
MSDM	16.21	17.47	12.71	13.29	14.92
MSDM (correction)	17.12	18.68	15.38	14.73	16.48
Gaussian Likelihood (Jayaram & Thickstun, 2020)					
ISDM	13.48	18.09	11.93	11.17	13.67
ISDM (correction)	14.27	19.10	12.74	12.20	14.58
MSDM	12.53	16.82	12.98	9.29	12.90
MSDM (correction)	13.93	17.92	14.19	12.11	14.54

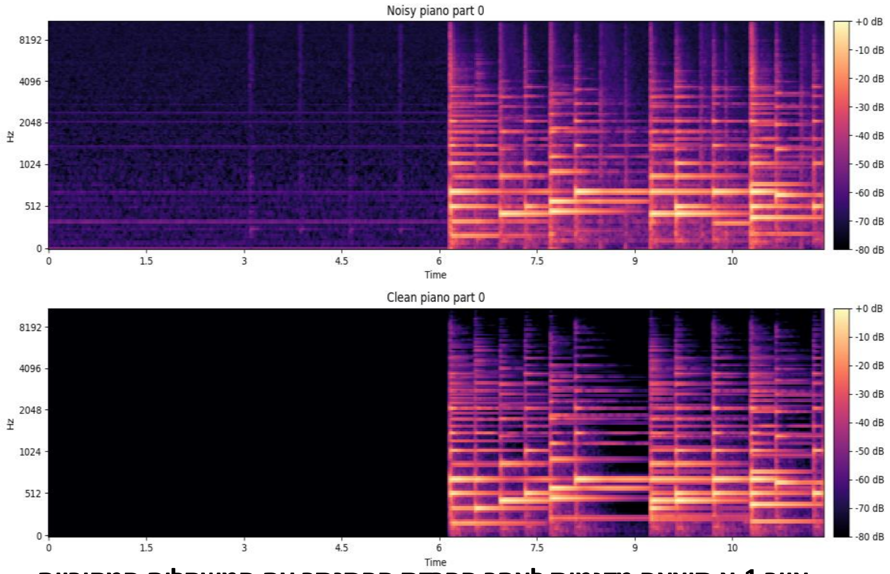
אנו כאמור השתמשנו במודל MSDM (correction) בשיטת Dirac Likelihood. גרסת correction משמעותה שימוש בטכניקת שיפור, בה בכל שלב i בדיפוזיה, מבצעים R איטרציות של הוספת רעש, ומבצעים אופטימיזציה מחדש (re-optimization) כאשר σ_i מקובע.

על מנת להתחיל מנקודת בסיס, כך שנוכל להשוות את התוצאות באופן יחסי לתוצאות אחרות, עשינו שתי ריצות בקרה לראות את הסטייה של המודל:

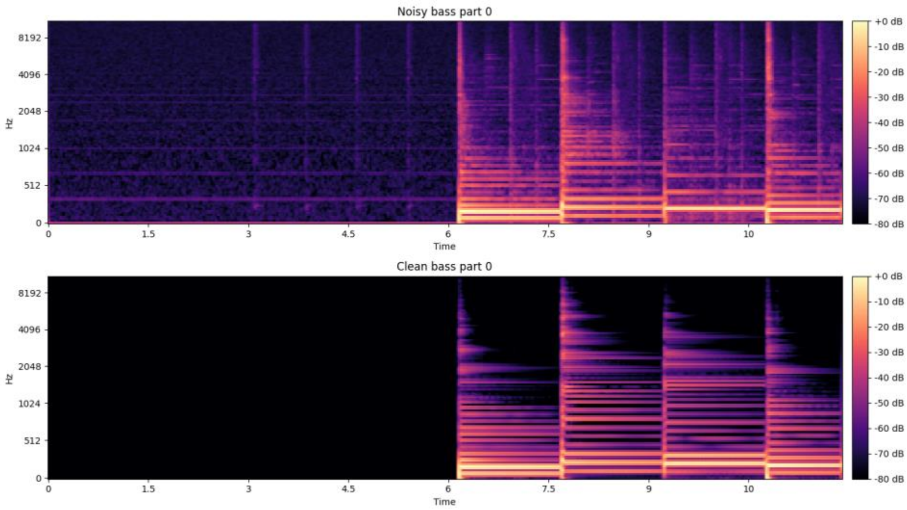


ניתן לראות כי יש שינוי אך לא מאוד גדול. בנוסף מבחינה איכותית (שמיעה) אין הבדל כלל.

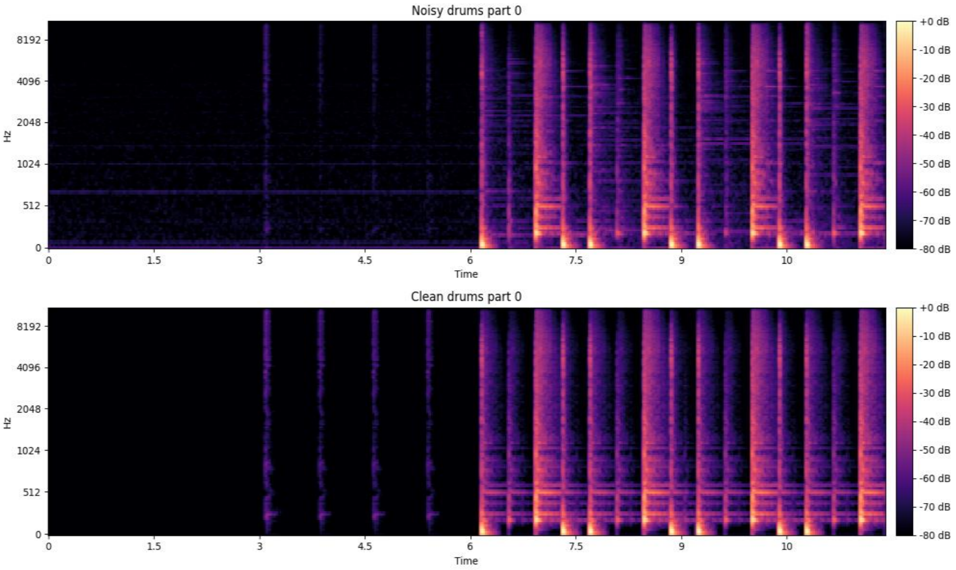
תוצאות evaluation עם משקלים מקוריים



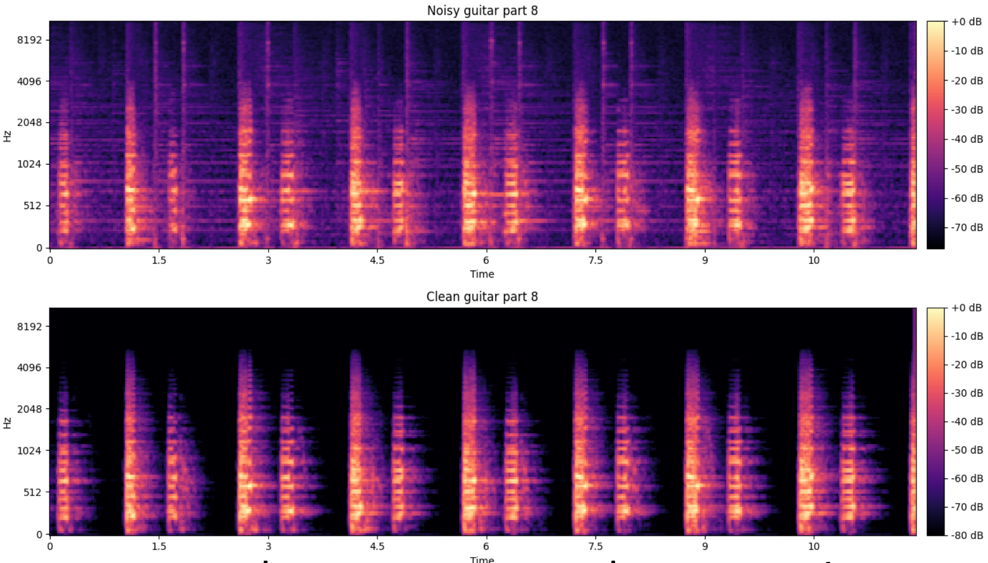
איור 1.א תוצאה מדגמית לאחר הפרדת הפסנתר עם המשקלים המקוריים



איור 1.ב תוצאה מדגמית לאחר הפרדת הבס עם המשקלים המקוריים



איור 1.ג תוצאה מדגמית לאחר הפרדת התופים עם המשקלים המקוריים



איור 1.ד תוצאה מדגמית לאחר הפרדת הגיטרה עם המשקלים המקוריים

ניתן לראות כי עבור כל הכלים התווספו תדרים בין 40dB ל-70dB. למעט התופים, לכל שאר הכלים תדרי התופים התווספו בעוצמה מסויימת. נסתכל על הבחינה הכמותית:

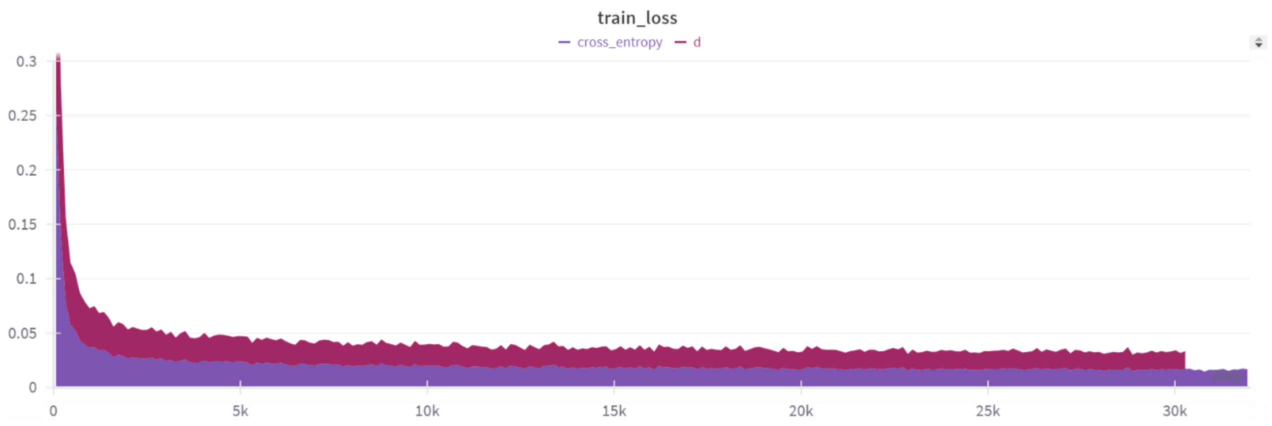
		Bass	Drums	Guitar	Piano	All
	Using given checkpoints	16.37	18.83	18.07	14.96	17.05
	Article	17.12	18.86	15.38	14.73	16.48
Train + Test	Training without modifications #1	14.43	16.18	18.91	14.81	16.08
	Training without modifications #2	14.75	16.89	18.90	14.40	16.23

טבלה 1. השוואה בין ריצות המקור - מאמר והגיט, לריצות הבקרה

ניתן לראות כי תוצאות הבקרה היו שונות בחלק מהכלים, אך היה לנו חשוב לעשות את הבקרה כדי לראות את היחסיות כאשר ניגש לשיפור התוצאות. כאשר בחנו את התוצאות מבחינה איכותית, ההבדל היה מזערי. ההפרדה הייתה בסך הכל טובה אבל לא מושלמת, ישנן "שאריות" כלים בכל כלי, מלבד התופים שהוא באופן יחסי ההפרדה שלו הייתה די טובה בכל אחת מהריצות.

<u>שינויי פונקציית Loss</u>

עבור שיטת שינוי הLoss מMSE ל-CCE קיבלנו:

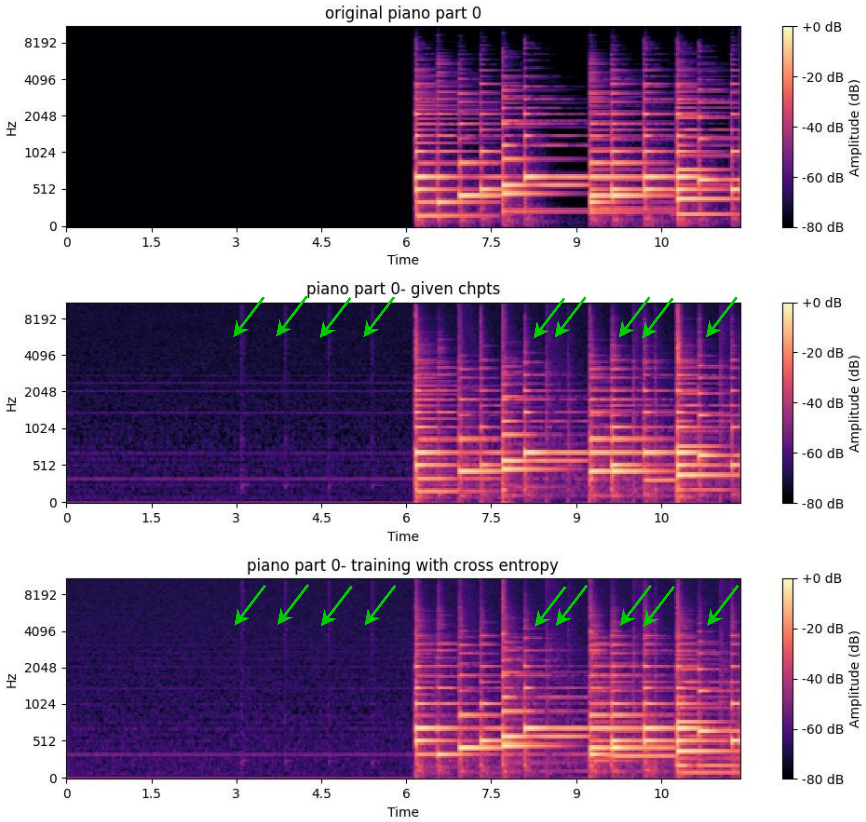


ניתן לראות כי ערכי הLoss של הCCE נמוכים לאורך הריצה מאשר ריצת הבקרה. מבחינה כמותית:

		Bass	Drums	Guitar	Piano	All
	Using given checkpoints	16.37	18.83	18.07	14.96	17.05
	Article	17.12	18.86	15.38	14.73	16.48
Train + Test	Training without modifications #1	14.43	16.18	18.91	14.81	16.08
	Training without modifications #2	14.75	16.89	18.90	14.40	16.23
	Training with modification: loss	17.03	19.11	18.43	15.54	17.53

טבלה 2. השוואה בין ריצות המקור לריצת שינוי הloss

ניתן לראות שבאופן מובהק שינוי פונקציית הLoss לCCE שיפרה משמעותית את תוצאות המודל. מבחינה איכותית, השינוי בשמע היה טוב יותר, אך לא בצורה יוצאת דופן.



ניתן לראות כי התדרים שהתווספו לאחר המעבר הראשון פחתו בעוצמתם כאשר שינינו את פונקציית הLoss.

<u>שינוי אופטימייזר מADAM ל- ADAMW</u>

		Bass	**Drums**	**Guitar**	**Piano**	**All**
	Using given checkpoints	16.37	18.83	18.07	14.96	17.05
	Article	**17.12**	18.86	<u>15.38</u>	14.73	16.48
Train + Test	Training without modifications #1	14.43	16.18	**18.91**	14.81	16.08
	Training without modifications #2	14.75	16.89	18.90	14.40	16.23
	Training with modification: loss	17.03	**19.11**	18.43	**15.54**	**17.53**
	Training with modification: optimizer	16.22	18.31	12.52	10.64	14.42

ניתן לראות כי מבחינה כמותית, אין שיפור בשינוי האופטימייזר ואף גרם לתוצאות הנמוכות ביותר בגיטרה ובפסנתר. מבחינה איכותית אין כמעט שינוי, מעט יותר "מלוכלך" וסינתטי אך

טבלה 3. השוואה בין ריצות המקור לריצת שינוי הloss לשינוי אופטימייזר ADAMW

די דומה למקור ולתוצאות שלאחר שינוי פונקציית הLoss בלבד.

<u>התמודדות המודל עם דאטא חדש</u>

על מנת לבחון את המודל עוד, יצרנו יצירה חדשה בעזרת תוכנת soundtrap, עם הכלים פסנתר, בס, גיטרה ותופים.[14] הכנסנו את כל הכלים לtest עם המשקלים המקוריים של הרשת. התוצאות שקיבלנו היו לא כמצופה.

---

[14] מיקס - https://drive.google.com/file/d/18FUO6SCLgo7ApbKbywvJbsCu05EOEG7C/view?usp=drive_link
בס - https://drive.google.com/file/d/1TxUFfYVCbW66mCLonbrmCcycyv5OVdV0/view?usp=sharing
תופים - https://drive.google.com/file/d/1_JHXaF033Zns_oTS5vLAkI4QDMoVald1/view?usp=sharing
גיטרה - https://drive.google.com/file/d/1P1ri5DXoNT71w2Q3a8ejhVMDK4dALpJm/view?usp=sharing
פסנתר - https://drive.google.com/file/d/1wU5fWggQNIR1rrDWvgdMVWqJVRRICiTT/view?usp=sharing

Instrument	SI-SDR
Bass	-5.5043
Drums	4.0051
Guitar	-8.96031
Piano	4.7785
All	-1.420

ייתכן שהמודל דורש תנאים מסויימים על מנת שיצליח להפריד כמו שצריך, לאחר בחינה ובדיקה של הדאטא החדש, לא ראינו הבדל גדול בין הדאטא המקורי לנוכחי.

עשינו ניסיון נוסף של הכנסת הדאטא DSD100, בס, גיטרה ותופים (פסנתר הוכנס כ"שקט" באורך המתאים). המודל שוב "תפקד" בצורה לא טובה כאשר הכנסנו דאטא חדש.

Instrument	SI-SDR
Bass	2.6454
Drums	3.3701
Guitar	5.8027
All	3.939

<u>התמודדות המודל עם שימוש חלקי בדאטא</u>

תחילה הכנסנו שני מקורות בלבד. לאחר מכן בחנו מה קורה כשיש 3 מקורות

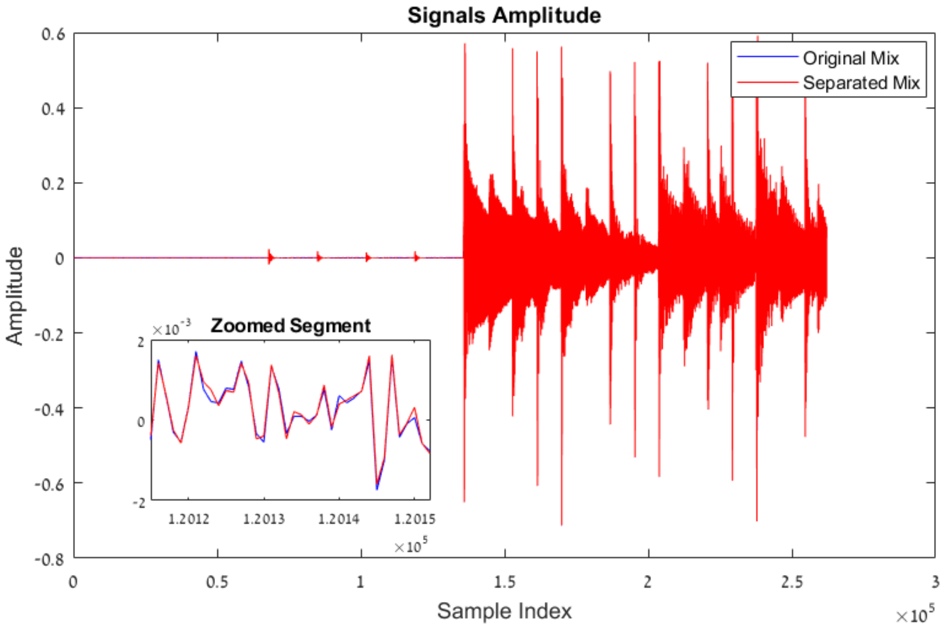
	Bass	Drums	Guitar	Piano	All
Article	17.12	18.86	15.38	14.73	16.48
Using given checkpoints	16.37	18.83	18.07	14.96	17.05
Using given checkpoints: only 2 sources -piano & drums	---	16.66	---	13.37	15.01
Using given checkpoints: only 2 sources -piano & bass	12.79	---	---	14.35	13.57
Using given checkpoints: only 3 sources -piano, bass & drums	15.92	18.34	---	15.09	16.45

ניתן לראות שברוב המקרים התוצאות היו פחות טובות, מלבד כאשר הכנסנו 3 מקורות, מדד הSI-SDR של הפסנתר היה הטוב ביותר לעומת כל המקרים שבחנו.

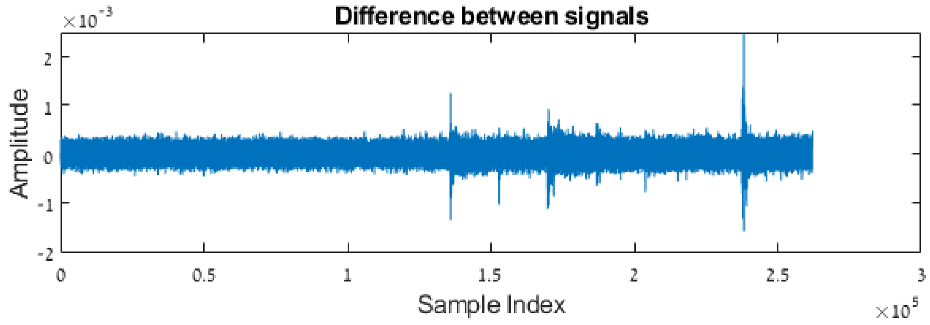
ייתכן כי המודל עובד בצורה מיטבית עבור 4 כלים, וככל שנפחית את מספר הכלים כך התוצאות יהיו יותר נמוכות.

<u>אופן השימוש בקלט</u>

בבחינה נוספת של המודל, בדקנו את סיגנל ה-mix המקורי אל מול סיגנל ה-mix המופרד, כאשר כל אחד מהם התקבל ע"י סכימת ארבעת הכלים (המקוריים/ המופרדים) לסיגנל אחד. מטרת הבחינה היתה לבדוק האם המודל מפריד את הקלט במלואו, או שהוא מוסיף/ מחסיר מרכיבים. להלן השוואה בין שני הסיגנלים שהתקבלו וכן תקריב של שני הגרפים:



להלן ההפרש בין שני הסיגנלים:



הסיגנלים שהתקבלו חופפים כמעט לחלוטין. ניתן להסיק מכך שהמודל משתמש בקלט במלואו, ומפריד אותו לארבעה כלים כמיטב יכולתו. תכונה זו אינה מובנת מאליה, הן בגלל האופי הסטוכסטי של מודל הדיפוזיה אשר מכיל רנדומיות רבה, והן מתוך ההנחה שמודל הדיפוזיה הבסיסי מתוכנן לבצע גנרציה של כלי בודד מתוך רעש.

סיכום ומסקנות

עבור השיפורים שעשינו, שינוי פונקציית הLoss הביאה את התוצאות הטובות ביותר. ניתן להסיק כי התאמת הפונקציית לסוג התהליך ואופיו גרמה לשיפור התוצאה. שינוי האופטימייזר לא תרם לשיפור התוצאות.

עבור הכנסת דאטא חדש, המסקנה שלנו היא שהמודל אינו יודע להתמודד עם דאטא חדש ולכן לוקה בחסר.

עבור הכנסה של דאטא חלקי, המודל לא התמודד בצורה מיטבית ולכן אנחנו מסיקים כי המודל אינו מאומן עבור דאטא חלקי, ופועל בצורה מיטבית עבור ארבעה כלים.

נספחים

בקישור הבא ניתן למצוא קבצי אודיו מתוך הפרויקט:

https://drive.google.com/drive/u/3/folders/1bbY81XQ3aDIh0ry-jMbcTt9I6VJxLMO0

תיקיית **original input** מכילה את ארבעת הכלים שה-mix ביניהם מהווה את הסיגנל אותו רצינו להפריד.

תיקיית **output_original_checkpoints** מכילה את פלט הרשת (כלים מופרדים) עבור ביצוע test עם המשקלים המקוריים הנתונים.

תיקיית **output_first_train** מכילה את פלט הרשת עבור אימון הרשת וביצוע test ללא שינויים כלשהם (אימון בקרה ראשון).

תיקיית **output_second_train** מכילה את פלט הרשת עבור אימון הרשת וביצוע test ללא שינויים כלשהם (אימון בקרה שני).

תיקיית **output_cross_entropy** מכילה את פלט הרשת עבור אימון הרשת וביצוע test כאשר פונקציית ה-Loss היא Cross Entropy.

תיקיית **output_second_pass_through_the_net** מכילה את פלט הרשת עבור ביצוע test, וביצוע test על תוצאות ה-test הקודם, כלומר עבור מעבר שני ברשת.

בתיקיות "...output_" (למעט תיקיית output_second_pass_through_the_net), בתוך תיקיית msdm-dirac ישנן 26 תיקיות. בכל תיקיה קיימים 4 הכלים שהופרדו, עבור מקטע זמן קטן (מקטע הקלט חולק ל-26 מקטעים).

בתיקיית "output_second_pass_through_the_net" ישנה תיקיה לכל כלי שאת הפרדתו רצינו לשפר. בכל כלי התיקיה special מכילה את 26 תיקיות. בכל תיקיה ישנה תיקיה "msdm-dirac/0", המכילה את 4 הכלים שהופרדו, עבור אותו מקטע זמן.

יצירת תמונות המכילות השוואת ספקטרוגרמות (STFT):

```
#  spectrograms

import os
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.gridspec import GridSpec

instrument = "piano"

# Define the path to your main directory and the path to the long file
main_dir = "D:/EngProg/denoise/msdm-dirac"
long_file_path = f"D:/EngProg/denoise/{instrument}.wav"
output_dir = f"D:/EngProg/denoise/spectrograms/{instrument}"

# Create the output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Load the long audio file
long_audio, sr_long = librosa.load(long_file_path, sr=None)

# Function to create spectrogram and plot it
def plot_spectrogram(y, sr, ax, title):
    S = librosa.feature.melspectrogram(y=y, sr=sr)
    S_dB = librosa.power_to_db(S, ref=np.max)
    img = librosa.display.specshow(S_dB, sr=sr, x_axis='time',
y_axis='mel', ax=ax)
    ax.set_title(title)
    return img

# Loop through the folders and files
for folder in range(26):  # From 0 to 25
    folder_path = os.path.join(main_dir, str(folder))

    # Load the corresponding short segment
    file_path = os.path.join(folder_path, f"{instrument}.wav")
    short_audio, sr_short = librosa.load(file_path, sr=None)

    # Calculate the starting time for the long file segment (in seconds)
    start_time = folder * 11.888616780045352

    # Get the length of the short audio in seconds
    short_duration = librosa.get_duration(y=short_audio, sr=sr_short)

    # Calculate the end time based on the short audio duration
    end_time = start_time + short_duration

    # Extract the corresponding part from the long file
    start_sample = int(start_time * sr_long)
    end_sample = int(end_time * sr_long)

    # Ensure the extracted segment does not exceed the length of the long
audio
    long_segment = long_audio[start_sample:end_sample]

    # Create a figure with a GridSpec layout
```

```
    fig = plt.figure(figsize=(14, 8))  # Increase the overall figure width
    gs = GridSpec(2, 2, width_ratios=[1, 0.02])  # Make the colorbar
thinner

    # Plot the short segment's spectrogram
    ax1 = fig.add_subplot(gs[0, 0])
    img1 = plot_spectrogram(short_audio, sr_short, ax1, f'Noisy
{instrument} part {folder}')

    # Plot the corresponding long file's spectrogram
    ax2 = fig.add_subplot(gs[1, 0])
    img2 = plot_spectrogram(long_segment, sr_long, ax2, f'Clean
{instrument} part {folder}')

    # Add colorbar for both spectrograms
    cbar1 = plt.colorbar(img1, cax=fig.add_subplot(gs[0, 1]),
format='%+2.0f dB', shrink=0.8)
    cbar2 = plt.colorbar(img2, cax=fig.add_subplot(gs[1, 1]),
format='%+2.0f dB', shrink=0.8)

    # Adjust layout to minimize space around subplots
    plt.subplots_adjust(hspace=0.3)  # Adjust vertical space between
subplots

    # Save the figure as an image
    output_img_path = os.path.join(output_dir, f"segment_{folder}.png")
    plt.tight_layout()
    plt.savefig(output_img_path, bbox_inches='tight')  # Use bbox_inches to
minimize extra space
    plt.close()

    print(f"Saved spectrogram image for segment {folder}")
```

מעבר שני ברשת:

```python
from pathlib import Path
import hydra
from omegaconf import DictConfig

from evaluation.evaluate_separation import evaluate_separations
from main import utils
import os
import shutil



log = utils.get_logger(__name__)


@hydra.main(config_path=".", config_name="config.yaml", version_base=None)
def main(config: DictConfig):

  # Define source and target directories
  source_base_dir = "/home/dsi/shremha2/multi-source-diffusion-models_retesting/data_to_sec/"
  target_dir = "/home/dsi/shremha2/multi-source-diffusion-models_retesting/data/slakh2100/test/Track01888/"
  separations_dir = "/home/dsi/shremha2/multi-source-diffusion-models_retesting/output/separations/"
  special_base_dir = "/home/dsi/shremha2/multi-source-diffusion-models_retesting/special/"

  for i in range(26):
    source_folder = os.path.join(source_base_dir, str(i))

    if os.path.exists(source_folder):
      # Clear the target directory before copying new content
      if os.path.exists(target_dir):
        shutil.rmtree(target_dir)  # Remove all files in target_dir
      os.makedirs(target_dir)  # Recreate target directory

      # Copy all content from source folder to target directory
      shutil.copytree(source_folder, target_dir, dirs_exist_ok=True)

    dataset_path = Path(config.dataset_path)
    separation_dir = Path(config.separation_dir)

    # Separate dataset
    separation_fn = hydra.utils.instantiate(config.separation)
    separation_fn(output_dir=separation_dir, dataset_path=dataset_path)

    # Compute metrics
    results = evaluate_separations(separation_dir, dataset_path, 22050, eps=1e-8)

    # Store and show results
    #results.to_csv(separation_dir/"metrics.csv")
    #log.info(f'Results:\n{results[["bass","drums","guitar","piano"]].mean()}')

    # Create a folder in /special/ named after 'i'
    special_folder = os.path.join(special_base_dir, str(i))
    os.makedirs(special_folder, exist_ok=True)  # Create the folder if it doesn't exist

    # Move the contents from separations/ to special/i/
    if os.path.exists(separations_dir):
        for filename in os.listdir(separations_dir):
            file_path = os.path.join(separations_dir, filename)
            shutil.move(file_path, special_folder)


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

חישוב SDR_i – SI עבור מעבר שני ברשת:

```
import torch
import torchaudio
from pathlib import Path

instrument = "bass"

# Function to calculate SI-SNR
def sisnr(preds: torch.Tensor, target: torch.Tensor, eps: float = 1e-8) ->
torch.Tensor:
    alpha = (torch.sum(preds * target, dim=-1, keepdim=True) + eps) / (
            torch.sum(target ** 2, dim=-1, keepdim=True) + eps)
    target_scaled = alpha * target
    noise = target_scaled - preds
    s_target = torch.sum(target_scaled ** 2, dim=-1) + eps
    s_error = torch.sum(noise ** 2, dim=-1) + eps
    return 10 * torch.log10(s_target / s_error)


# Function to load and resample audio
def load_audio(file_path: Path, target_sr: int = 22050) -> torch.Tensor:
    wav, sr = torchaudio.load(file_path)
    if sr != target_sr:
        wav = torchaudio.functional.resample(wav, orig_freq=sr,
new_freq=target_sr)
    return wav, sr


# Function to calculate SI-SNR improvement for one pair of files
def calculate_sisnr_improvement(original_segment: torch.Tensor,
mixture_segment: torch.Tensor, separated_file: str,
                                resample_sr: int = 22050):
    s, sr_s = load_audio(separated_file, resample_sr)

    # Ensure all audios have the same sample rate
    assert sr_s == resample_sr, "Sample rates do not match!"

    # Ensure all files are the same length
    min_len = min(original_segment.shape[1], s.shape[1],
mixture_segment.shape[1])
    original_segment = original_segment[:, :min_len]
    s = s[:, :min_len]
    mixture_segment = mixture_segment[:, :min_len]

    # Compute SI-SNR improvement
    sisnr_separated = sisnr(s, original_segment)
    sisnr_mixture = sisnr(mixture_segment, original_segment)
    sisnr_improvement = (sisnr_separated - sisnr_mixture).item()

    return sisnr_improvement


# Function to process all "piano" files in the piano_out folder
def calculate_sisnr_for_all_pianos(piano_out_dir: str, original_file: str,
mixture_file: str, resample_sr: int = 22050,
                                   segment_duration: float =
11.888616780045352):
    piano_out_dir = Path(piano_out_dir)
```

```
    sisnr_improvements = []

    # Load the original long audio file and mixture file
    original_audio, sr_o = load_audio(original_file, resample_sr)
    mixture_audio, sr_m = load_audio(mixture_file, resample_sr)

    assert sr_o == sr_m == resample_sr, "Sample rates for original and
mixture do not match!"

    total_duration_original = original_audio.shape[1] / sr_o  # Total
duration in seconds for original
    total_duration_mix = mixture_audio.shape[1] / sr_m  # Total duration in
seconds for mix

    # Calculate the number of segments based on the original audio duration
    num_segments = int(total_duration_original / segment_duration)

    # Iterate through each segment
    for segment_idx in range(num_segments):
        # Calculate start and end time for this segment
        start_time = segment_idx * segment_duration
        end_time = (segment_idx + 1) * segment_duration

        # Convert time to samples
        start_sample = int(start_time * sr_o)
        end_sample = int(end_time * sr_o)

        # Ensure we don't exceed the length of the original audio
        original_segment = original_audio[:, start_sample:end_sample]
        mixture_segment = mixture_audio[:, start_sample:end_sample]

        # Update the path for the separated file based on the segment index
        separated_file =
Path(f"D:/EngProg/denoise/all_second_clean/{instrument}_out/special/{segmen
t_idx}/msdm-dirac/0/{instrument}.wav")

        if separated_file.exists():
            sisnr_improvement =
calculate_sisnr_improvement(original_segment, mixture_segment,
separated_file,
                                                            resample_sr)
            sisnr_improvements.append(sisnr_improvement)
            print(f"SI-SNR Improvement for {separated_file}:
{sisnr_improvement:.5f} dB")

    # Compute and print average SI-SNR improvement
    if sisnr_improvements:
        average_sisnr = sum(sisnr_improvements) / len(sisnr_improvements)
        print(f"Average SI-SNR Improvement: {average_sisnr:.5f} dB")
    else:
        print("No 'piano.wav' files found.")

# Example usage
net_out_dir = f"D:/EngProg/denoise/all_second_clean/{instrument}_out"
original_file =
f"D:/EngProg/denoise/all_second_clean/full_original/{instrument}.wav"
mixture_file = "D:/EngProg/denoise/all_second_clean/mix.wav"

calculate_sisnr_for_all_pianos(net_out_dir, original_file, mixture_file)
```

חישוב שגיאה ממוצעת עבור שיטת שיפור:

```
# 3 spectrograms and mean error
import os
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.gridspec import GridSpec

# Define the path to your main directory and the path to the long file
instrument = "bass"
main_dir = "D:/EngProg/denoise/msdm-dirac"
long_file_path = f"D:/EngProg/denoise/{instrument}.wav"
output_dir = f"D:/EngProg/denoise/msdm-dirac_cross_entropy/{instrument}"
third_spec_base_path = f"D:/EngProg/denoise/msdm-dirac_cross_entropy"

# Create the output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Load the long audio file
long_audio, sr_long = librosa.load(long_file_path, sr=None)

# Function to create spectrogram and plot it
def plot_spectrogram(y, sr, ax, title):
    S = librosa.feature.melspectrogram(y=y, sr=sr)
    S_dB = librosa.power_to_db(S, ref=np.max)
    img = librosa.display.specshow(S_dB, sr=sr, x_axis='time',
y_axis='mel', ax=ax)
    ax.set_title(title)
    return S_dB, img

# Function to calculate spectrogram error based on frequency
def calculate_spectrogram_error(S1, S2):
    min_width = min(S1.shape[1], S2.shape[1])
    S1_trimmed = S1[:, :min_width]
    S2_trimmed = S2[:, :min_width]
    error = np.mean(np.abs(S1_trimmed - S2_trimmed), axis=1)
    return error

# Initialize list to store errors for all segments
all_errors = []

# Loop through the folders and files
for folder in range(26):  # From 0 to 25
    folder_path = os.path.join(main_dir, str(folder))

    # Load the corresponding short segment
    file_path = os.path.join(folder_path, f"{instrument}.wav")
    short_audio, sr_short = librosa.load(file_path, sr=None)

    # Calculate the starting time for the long file segment (in seconds)
    start_time = folder * 11.888616780045352

    # Get the length of the short audio in seconds
    short_duration = librosa.get_duration(y=short_audio, sr=sr_short)

    # Calculate the end time based on the short audio duration
    end_time = start_time + short_duration
```

```
    # Extract the corresponding part from the long file
    start_sample = int(start_time * sr_long)
    end_sample = int(end_time * sr_long)

    # Ensure the extracted segment does not exceed the length of the long
audio
    long_segment = long_audio[start_sample:end_sample]

    # Load the additional spectrogram audio file
    third_spec_path = os.path.join(third_spec_base_path, str(folder),
f"{instrument}.wav")
    third_audio, sr_third = librosa.load(third_spec_path, sr=None)

    # Create a figure with a GridSpec layout for 3 spectrograms
    fig = plt.figure(figsize=(14, 12))  # Adjust the height for 3 rows
    gs = GridSpec(3, 2, width_ratios=[1, 0.02])  # Maintain a thinner
colorbar

    # Plot the long audio segment (S1) at the top
    ax1 = fig.add_subplot(gs[0, 0])
    S1, img1 = plot_spectrogram(long_segment, sr_long, ax1, f'S1: Clean
{instrument} part {folder}')

    # Plot the short audio segment (S2) in the middle
    ax2 = fig.add_subplot(gs[1, 0])
    S2, img2 = plot_spectrogram(short_audio, sr_short, ax2, f'S2: Short
{instrument} part {folder}')

    # Plot the third audio segment (S3) at the bottom
    ax3 = fig.add_subplot(gs[2, 0])
    S3, img3 = plot_spectrogram(third_audio, sr_third, ax3, f'S3: Noisy
{instrument} part {folder}')

    # Add colorbars for each spectrogram
    plt.colorbar(img1, cax=fig.add_subplot(gs[0, 1]), format='%+2.0f dB',
shrink=0.8)
    plt.colorbar(img2, cax=fig.add_subplot(gs[1, 1]), format='%+2.0f dB',
shrink=0.8)
    plt.colorbar(img3, cax=fig.add_subplot(gs[2, 1]), format='%+2.0f dB',
shrink=0.8)

    # Adjust layout to minimize space around subplots
    plt.subplots_adjust(hspace=0.4)

    # Save the figure as an image
    output_img_path = os.path.join(output_dir, f"segment_{folder}.png")
    plt.tight_layout()
    plt.savefig(output_img_path, bbox_inches='tight')
    plt.close()

    print(f"Saved spectrogram image for segment {folder}")

    # Calculate errors and store them
    error_S1_S2 = calculate_spectrogram_error(S1, S2)
    error_S1_S3 = calculate_spectrogram_error(S1, S3)

    # Store errors in a structured way
    all_errors.append({
```

```
        'error_S1_S2': error_S1_S2,
        'error_S1_S3': error_S1_S3
    })

# After processing all segments, create the final error graphs
freqs =
librosa.mel_frequencies(n_mels=all_errors[0]['error_S1_S2'].shape[0],
fmin=0, fmax=sr_long/2)

# Graph with mean error across all segments
fig, ax = plt.subplots(figsize=(8, 8))  # Set the figure to be square

mean_error_S1_S2 = np.mean([e['error_S1_S2'] for e in all_errors], axis=0)
mean_error_S1_S3 = np.mean([e['error_S1_S3'] for e in all_errors], axis=0)

# Plot Mean Errors
ax.plot(freqs, mean_error_S1_S2, label='Mean Error Clean-First',
color='blue')
ax.plot(freqs, mean_error_S1_S3, label='Mean Error Clean-Cross Entropy',
color='green')

ax.set_title(f'Mean Absolute Error Between {instrument.capitalize()}
Spectrograms - Average Across All Segments')
ax.set_xlabel('Frequency (Hz)')
ax.set_ylabel('Mean Absolute Error')
ax.set_xscale('log')

# Position the legend within the plot area, typically in the upper left
ax.legend(loc='upper left', fontsize='x-large', frameon=True)

plt.tight_layout()
plt.savefig(os.path.join(output_dir, "mean_error_segments.png"),
bbox_inches='tight')
plt.close()
```

התמרת FFT עבור הכלים המקוריים:

```
% Read audio files
[bass, fs_b] = audioread('bass.wav');
[drums, fs_d] = audioread('drums.wav');
[guitar, fs_g] = audioread('guitar.wav');
[piano, fs_p] = audioread('piano.wav');


% Compute the FFT of each signal
fft_bass = fft(bass);
fft_drums = fft(drums);
fft_guitar = fft(guitar);
fft_piano = fft(piano);


% Compute the frequency axes
f_b = (0:length(fft_bass)-1) * (fs_b / length(fft_bass));
f_d = (0:length(fft_drums)-1) * (fs_d / length(fft_drums));
f_g = (0:length(fft_guitar)-1) * (fs_g / length(fft_guitar));
f_p = (0:length(fft_piano)-1) * (fs_p / length(fft_piano));


% Magnitudes of the FFT (only positive frequencies)
m_bass = abs(fft_bass(1:floor(length(fft_bass)/2)));
m_drums = abs(fft_drums(1:floor(length(fft_drums)/2)));
m_guitar = abs(fft_guitar(1:floor(length(fft_guitar)/2)));
m_piano = abs(fft_piano(1:floor(length(fft_piano)/2)));


% Corresponding frequency vectors for positive frequencies
f_b = f_b(1:floor(length(f_b)/2));
f_d = f_d(1:floor(length(f_d)/2));
f_g = f_g(1:floor(length(f_g)/2));
f_p = f_p(1:floor(length(f_p)/2));


% Find the index corresponding to 1000 Hz in each frequency vector
idx_b = find(f_b >= 1000, 1);
idx_d = find(f_d >= 1000, 1);
idx_g = find(f_g >= 1000, 1);
idx_p = find(f_p >= 1000, 1);


% Plot all FFTs in two subplots
figure;


% Frequencies from 0 to 1000 Hz
subplot(2,1,1);
hold on;
plot(f_b(1:idx_b), m_bass(1:idx_b), 'b', 'DisplayName', 'Bass');
plot(f_d(1:idx_d), m_drums(1:idx_d), 'r', 'DisplayName', 'Drums');
plot(f_g(1:idx_g), m_guitar(1:idx_g), 'g', 'DisplayName', 'Guitar');
```

```
plot(f_p(1:idx_p), m_piano(1:idx_p), 'y', 'DisplayName', 'Piano');
hold off;
xlabel('Frequency (Hz)');
ylabel('Magnitude');
legend;
grid on;


% Frequencies above 1000 Hz
subplot(2,1,2);
hold on;
plot(f_b(idx_b+1:end), m_bass(idx_b+1:end), 'b', 'DisplayName',
'Bass');
plot(f_d(idx_d+1:end), m_drums(idx_d+1:end), 'r', 'DisplayName',
'Drums');
plot(f_g(idx_g+1:end), m_guitar(idx_g+1:end), 'g', 'DisplayName',
'Guitar');
plot(f_p(idx_p+1:end), m_piano(idx_p+1:end), 'y', 'DisplayName',
'Piano');
hold off;
xlabel('Frequency (Hz)');
ylabel('Magnitude');
legend;
grid on;


% Title for the whole figure
sgtitle('FFT of Bass, Drums, Guitar, and Piano');
```