

המחלקה להנדסת חשמל

שם הפרויקט: הפרדת כלי נגינה וזמר/ת  
מהקלטות של שירים.

Project Name: separation of musical  
instruments and singer recordings of  
songs.

ספר הפרוייקט

שם הסטודנט: בן ציון צוברי

מספר תעודת זהות:

שם המנחה: שגיא הרפז

חתימת המנחה:

תאריך ההגשה: 5.6.2021

## תודות:

תהליך ביצוע פרויקט הגמר היה לא פשוט כלל, הייתי צריך לשלב בין המסגרת הצבאית מרובת האחראיות והשגרה הלא צפויה לבין למידה של חומרים חדשים שלא נחשפתי אליהם לעומק וההתקדמות בפרויקט הגמר, לכן ברצוני להביע את הערכתי ותודתי לאנשים שעזרו לי במהלך הדרך.

- להורים שלי.
  - למנחה הפרויקט שגיא הרפז.
  - לניר אלנברג וסגל מחלקת פרויקטים ואפקה.
  - לחברים הקרובים ולחברים מהצבא.
- בלעדיהם לא הייתי מצליח להתקדם הלאה ולסיים את מטלת פרויקט הגמר.

## תוכן עניינים:

4.....	רשימות
5.....	תקציר
7.....	<i>Executive Summary</i>
9.....	מילון מונחים
10.....	מבוא
12.....	מטרת הפרויקט, יעדים ומדדים
13.....	מקירת ספרות
	ניתוח חלופות
14.....	חלופה מערכתית
15.....	חלופה טכנולוגית – GPU
	תכן מפורט
17.....	דרישות בסיסיות ליישום הפרויקט
18.....	רשת קונבולוציה
20.....	אופטימיזציה
21.....	ADAM Optimizer
21.....	MUSDB18 Dataset
22.....	Wave – U – Net
23.....	המודל של הפרויקט ותהליך האימון
27.....	חבילת הקוד של הפרויקט
29.....	הרצת תהליך האימון ושימוש במודל
32.....	התוצר
	בדיקות והערכה
33.....	MSE
34.....	SNR
35.....	MOS
36.....	STOI
37.....	סיכום ומסקנות
38.....	הצעות עבודה להמשך

39..... תכנון הפרויקט, ריכוז שינויים וניהול סיכונים

40..... רשימת מקורות

#### נספחים

41..... נספח א' – מפרט מערכת CPU + GPU

42..... נספח ב' – Python code

54..... נספח ג' – פורמט סקר MOS

55..... נספח ד' – פוסטר הפרויקט

## רשימות:

### איורים:

- איור 1: *Wave – U – Net*.
- איור 2: *McCulluch – Pitts Model*.
- איור 3: *2d – convolution*.
- איור 4: *psuedo code for ADAM*.
- איור 5: ארכיטקטורת *Wave – U – Net*.
- איור 6: דיאגרמת בלוקים לאלגוריתם מציאת המודל.
- איור 7: דיאגרמת בלוקים מופשטת ל- *Wave – U – Net*.
- איור 8: חבילת הקוד.
- איור 9: הרצת התוכנית.
- איור 10: מחזור *epoch* שלם.
- איור 11: בחינת אבלואציה סופית למודלים.
- איור 12: ביצוע הפרדה ב- *CMD*.

### גרפים:

- גרף 1: חישוב MSE של רכיב ה- *Vocals*.
- גרף 2: חישוב MSE של רכיב ה- *Drums*.
- גרף 3: חישוב MSE של רכיב ה- *Bass*.
- גרף 4: חישוב MSE של רכיב ה- *Other*.
- גרף 5: חישוב SNR להפרדות שירים מה- *Training set*.
- גרף 6: תוצאות מדד *STOI*.

### טבלאות:

- טבלה 1: ניתוח חלופות מערכתיות.
- טבלה 2: ניתוח חלופות טכנולוגיות.

### נוסחאות:

- נוסחא 1: חישוב *MSE*.
- נוסחא 2: חישוב *MOS*.
- נוסחא 3: *Gradient Descent*.
- נוסחא 4: *Difference output*.
- נוסחא 5: חישוב *SNR*.

## תקציר:

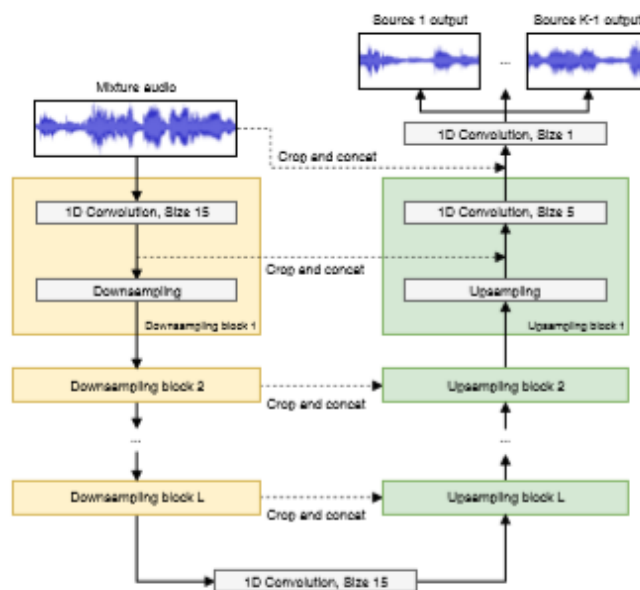
כיום, נפוץ שמשתמשים אינטרנטיים נעזרים בתוכנות ייעודיות על מנת להפריד שירים בפורמטים שונים למקורות (זמר/ת, בס תופים, צלילים אחרים), תוכנות אלו עולות כסף, לרוב לא נוחות לשימוש וצורכות זמן למידה של המשתמש, פרויקט זה עוסק בתהליך יישום מערכת קוד מבוסס Deep Learning הניתנת לאימון, שמטרתה לבצע את ההפרדה של השירים למקורות.

### מערכות Deep Learning:

- אוטומטיות וקלות לשימוש.
- מבצעות את המשימה במהירות.
- דורשות הרצה חד-פעמית של תהליך אימון למודל.
- קיימת אופציה ללמידה מונחית באמצעות Dataset.
- בעלות יכולת להתפתח עם הזמן.

המטרה העיקרית של הפרויקט, מעבר ליישום המערכת, היא להראות שניתן לפתור בעיות מורכבות כמו הפרדות של שירים למקורות באמצעות אלגוריתמי *Deep learning*.

במהלך הפרויקט ובדו"ח זה נבחן את ה-Wave – U – Net, שהוא אלגוריתם ללמידה עמוקה מסוג CNN (Convolutional Neural Network) המיועד לפתירת משימות של הפרדה למקורות במרחב הזמן.



איור 1: Wave-U-Net

ה-Wave – U – Net משתמש ב- *musdb18 dataset* המכיל שירים מקוריים ואת ההפרדות המקוריות שלהם שהוקלטו בערוצים נפרדים ובאמצעותם המודל "לומד" כיצד לבצע את ההפרדות, כל זה קורה בתהליך האימון של המערכת ובסופו מתקבל מודל סופי מבין הרבה שהוא בעל הביצועים הטובים ביותר להפרדות שירים [1].

בתהליך האימון ובניית מודל ה-Wave – U – Net השירים מסט האימון עוברים במודל במספר רב של איטרציות כאשר בכל אחת מהן נבחן מודל יחיד ובסופה מוחלט האם הוא טוב יותר או לא לפי ה-Loss Function שלו הנותנת את ההפסד הכולל על בחינת שירים מסט הולידציה של ה-Dataset.

כל מודל מכיל  $L$  שכבות של  $CL$  (Convolution Layers) שתפקידן לקבל את ה-input מהשכבה הקודמת ולבצע  $1d$  Convolution בין המידע בכניסה לשכבה למספר "פילטרים" (נקראים *Kernels*) ליצירת *Feature Maps* שהן המרכיב הכי חשוב של המודל שנבנה ובעזרתן מתבצעים ההפרדות. ו-Up/Down sampling בין כל  $CL$ , סה"כ  $2L$  שכבות, כדי לחשב *Feature Maps* נוספים ברזולוציות זמן שונות.

לאחר בניית המודל המפריד, הוא עובר תהליך של Optimization שבו אנו נרצה לקבל את ערכי המשקלים ב- *Kernels* עבורם ה-Loss Function של המודל מינימלית וה- *Feature Maps* שמחושבים באמצעותם אופטימליים, נשתמש ב- *ADAM Optimizer* עם מקדם *LR* (Learning Rate) של 0.0001 כדי להמנע מ- *Over/Under Fitting* ולוודא את התכנסות המודל.

על מנת לשפר את המודל יותר מתבצע תהליך אימון שני מייד בסיום החלק הראשון הנקרא *Fine Tuning* שבו מקשיחים את התנאים לאימון המודל ע"י כך שמתבצעים צעדים קטנים יותר בכיוון המינימום של ה-Loss Function וממשיכים בביצוע סט איטרציות נוסף של אימון למודל.

המודל הסופי הוא התוצר של תהליך האימון המשני ונשמר על המחשב בתיקיית הקוד, מודל זה מכיל את הפרמטרים והמשקלים האופטימליים לביצוע הפרדות שירים למקורות בהתאם ל- *musdb18 dataset* ובו נשתמש בביצוע הפרדות לשירים.

לתוצרי המודל הסופי שהתקבל בסוף התהליך בוצעו מספר בדיקות והערכות לפי מדדי *MSE, SNR, MOS, STOI* המתארים את איכות, מובנות וחווית משתמש של תוצרי ההפרדות של המודל, התוצאות נדגמו משירים שהופרדו באופן עצמי מסט האימון של הפרויקט בהשוואה לקבצי המקור. ביצועי המודל שנבחן הראו על תוצאות טובות עם יותר מ-  $10\text{ dB}$  במדד *SNR*,  $70\%$  התאמה במדד *STOI*, ממוצע של מעל 7.5 במדד *MOS* ו- *MSE* מינימלי.

פרויקט זה מציג את יכולותיהם של אלגוריתמי למידה עמוקה ויישומן בפתירת בעיות מורכבות כמו הפרדות של שירים למקורות ביעילות ובהצלחה.

## Executive Summary:

Nowadays, it is common for personal internet users to use different types of software's to separate songs to their multi-instrument factors (vocals, bass, drums and other sounds), these software's require paid subscription for usage and are hard to use and need time to study them. This project suggests a code based platform using trainable Deep Learning methods to accomplish a fully end-to-end song source separation.

Deep learning systems:

- Are automatic and easy to use.
- Accomplishes the task quickly.
- Require a 1 time only training procedure for the model.
- Supervised learning can be applied using labeled dataset.
- Evolves through experience.

The main goal of this project is to show that complex problems such as multi-instrument song separation can be solved using Deep learning algorithms.

During the project we will be using the Wave-U-Net, a multi-scale end-to-end CNN (Convolutional Neural Network) that operates on the time-domain and separates to song to its individual sources.

The Wave-U-Net uses musdb18 dataset that includes original songs with their respective multi-instrument separations for the training procedure of the system model [1].

In the training procedure the files from the dataset are loaded in batches into the Wave-U-Net to create a parametric model of the separator, this procedure keeps going while the system is learning and creating many different models that are being testes and evaluated in accordance to their total loss on song prediction from the validation dataset.

Each model consists on L blocks of convolution layers and down/up sampling operation. CL performs the 1dConvolution operation between their input and a number of randomly assigned filters (kernels) to compute feature maps for the model, these feature maps are being used by the model to make the song predictions after being optimized. Down/up sampling is performed between each CL, L layers each for 2L total layers, to calculate different feature maps in different resolutions.

The separator model calculated in the current iteration is going through an optimization procedure where we want to minimize the loss function in accordance to the kernel weights, we use ADAM optimizer algorithm for optimization and learning rate factor 0.0001 to avoid over/under fitting of the model.

To further enhance the separator model performance fine tuning is applied after the first set of training, the learning rate is lowered to 0.00001 as we take smaller steps in the minimum direction of the loss function and carry out another set of training.



The final model is the product of the secondary training set and is saved on the PC. We use it to make our song separation prediction because this model holds the optimized parameters and weights for multi-instrument source separation given the musdb18 dataset.

We perform some tests and evaluations on the source estimates final separator model created in the training process, the indices used are SNR and MSE for quality, STOI for intelligibility and MOS for user experience. The results for the calculations of these indices were based on separations from the training set and their references. Our models performance were very good and showed successful results on all indices with over 10 dB on SNR on more than half the songs testes, 70% on most of the songs with STOI, an average score over 7.5 rating on MOS and minimal MSE.

This project showed the capabilities of deep learning algorithms and that they can solve complex problems through training such as multi-instrument song source separation with great results

## מילון מונחים:

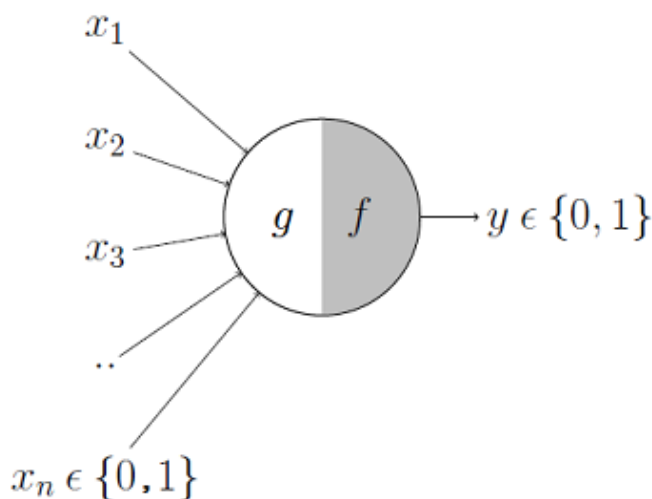
1. Machine learning - הוא תת-תחום בעולם הבינה המלאכותית המתאר אלגוריתמים אשר מסוגלים ללמוד לפתרון בעיות מסוגים שונים בהינתן *Dataset* ידוע מראש.
2. Deep learning - הוא תת-תחום של *Machine learning* המניח שניתן לדמות את פעולת המוח האנושי באמצעות אלגוריתמים לומדים.
3. State – of – the – Art - משמש המונח לתיאור האלגוריתם העדכני הטוב ביותר בהיבטי ביצועים לפתרון הבעיה הנידונה.
4. Gradient - אופרטור וקטורי המתאר את הנגזרת של פונקציה בעלת מספר משתנים.
5. Back Propagation - אלגוריתם לעדכון משקלי הרשת באמצעות ה- *Gradient* של ה- *Loss function* לרשת.
6. Feature maps - תוצרי שכבת הקונבולוציה שבעזרתן המערכת הלומדת מבצעת פרדיקציות בהתאם לאות המידע.
7. Learning rate - פרמטר בקונפיגורציה המודל האחראי על ויסות קצת הלמידה של המערכת.
8. SNR – Signal to Noise Ratio - יחס אות לרעש, מדד כמותי שבאמצעותו ניתן לתאר כמה רעש נכנס בתוך קטע האודיו הנדגם.
9. MSE (Mean Squared Error) - מדד כמותי המתאר את ממוצע ריבוי השגיאות בין אומד מסויים למה שנאמד.
10. MOS – (Mean Opinion Score) - מדד כמותי לחווית משתמש על בסיס דירוג התוצר.
11. STOI – Short Term Object Inteligibility - מדד איכות לרמת המובנות של קטע אודיו.

## מבוא:

כל שיר מורכב מ-4 מקורות: זמר/ת, תופים, בס וצלילים אחרים (כמו גיטרות, פסנתר, חצוצרות וכו'). לרוב הפרדות של שירים למקורות אלו מתבצעות באופן ידני באמצעות תוכנות ייעודיות לעריכת אודיו, חלק מתוכנות אלו דורשות מנוי חודשי לשימוש באפליקציה ויש ללמוד כיצד לעבוד עם הכלים של כל תוכנה כאשר החלק הנותר מפיק תוצאות שאינן איכותיות.

פרויקט זה עוסק בתכנון ויישום מערכת אוטומטית המפרידה שירים למקורות באופן מיידי ואיכותי באמצעות אלגוריתמי *Deep Learning*, שכיום מיושמים במערכות *AI* והתפתחו רבות בשנים האחרונות.

*Deep Learning* היא תחום מחקר בעולם המחשבים וה-*AI* המתבססת על ההנחה שניתן לדמות באופן ממוחשב את פעולות הנירונים במוח וכך ליצור מערכות עם היכולת ללמוד ולהשתפר עם הזמן. הבסיס המתמטי להנחה זו הוצע ע"י *Walter Pitts* ו-*Warren McCulloch* כאשר הם יצרו את ה-*Threshold logic* באמצעות מודל שפועל באופן דומה (עפ"י מה שהיה ידוע אז) לנירון במח – נקרא גם *McCulloch – Pitts Model*.



איור 2: *McCulloch – Pitts Model*

המודל מקבל בכניסתו סדרה של מידע  $[X_1, X_2 \dots X_n]$  ומשקליו  $[W_1, W_2 \dots W_n]$  כך שלפי הפונקציה  $f(X, W) = \sum_{i=1}^n X_i W_i$  תתקבל תוצאה שהיא '1' או '0', את המשקלים היו מזינים באופן ידני.

רק יותר מאוחר נבנה ה-*Perceptron* ע"י *Frank Rosenblatt* שהיה מעדכן את המשקלים באופן אוטומטי אך לא זכה לתהילה רבה בשל ביצועים שלא עמדו בציפיות.

הנחת יסוד זאת הביאה לפיתוחם של אלגוריתמים נוספים כמו *Back Propagation* ב-*Machine Learning* שפותח בשנות ה-60 ע"י *Henry J. Kelley* אך יושם בפועל החל משנות ה-80 ו-*CNN (Convolutional Neural Networks)* הוצעה לראשונה ע"י *Kunihiko Fukushima*. אך בתקופה שבין שנות ה-70 לתחילת שנות ה-2000 רווחה אכזבה גדולה מהביצועים של אלגוריתמים אלו שלא עמדו בציפיות גם כן בשל אי עמידה של חומרה ביכולות החישוב וחוסר במשאבים.

החל מאמצע שנות ה-2000, החוקרים *Yoshua Bengio* ו- *Yan LeCun, Geoffery Hinton* הובילו סדרה של ניסויים כאשר תכליתם היה ליצור רשתות 'עמוקות' יותר ויותר. סדרת הניסויים שערכו הניבו תוצאות מעולות וזאת בזכות 2 דברים חשובים שקרו במהלך הזמן:

1. תחילת עידן האינטרנט ואיסוף המידע במאגרים וכווננים קשיחים, כמות המידע שהיה ניתן לצבור ולאמן באמצעותו את המערכות גדל אקספוננציאלית.
2. מחשבים ובעיקר חומרה השתפרו מאוד בביצועים ביכולות החישוב שלהם, עוצמת החישוב המשתפרת השפיעה באופן ישיר על הצלחות הניסויים.

כך החלו חוקרים רבים ללמוד את התחום וללמד מערכות עמוקות יותר ויותר ומכאן בא המונח *Deep Learning*.

כיום, עם מעבדי *GPU* חזקים ונגישים וכמות המידע העצומה שנצברה, הצליחו מערכות לומדות להשתלב במגוון תחומים מרכזיים בחיינו כמו רפואה, זיהוי עצמים ודיבור, הפרדת מקורות, נהיגה אוטונומית, מסחר ועוד.

בפרויקט זה נרצה לפתח מערכת לומדת לפתרון משימת הפרדת מקורות, באמצעות יישום ה- *Wave – U – Net*, המערכת תקבל בכניסתה קובץ אודיו המכיל שיר ובמוצאה תפיק 4 קבצים מתאימים למקורות המופרדים של השיר לזמר/ת, לתופים, לבס ולצלילים אחרים כאשר הגישה היא להשתמש באלגוריתמי *Deep Learning* הפועלים על השיר באופן ישיר.

## מטרת הפרויקט, יעדים ומדדים:

### מטרת הפרויקט:

תכנון ויישום מערכת לומדת שתקבל ב- *Input* קובץ אודיו, תבצע הפרדה של השיר למקורות ותפיק ב- *Output* קובץ מתאים עבור כל מקור מופרד – זמר/ת, תופים, בס, צלילים גבוהים.

### יעדי הפרויקט:

1. **היעד:** יכולת הפרדה ברזולוציה גבוהה עם מעט רעשים בסיגנל הרצוי, על מנת להבחין באופן ברור שקיימת הפרדה בין השיר למקור הרצוי באופן גורף על פני כל המקורות.

### המדדים:

- *SNR* מינימלי של  $10[dB]$  למקורות המשוערכים ב- *output* המערכת.
- *MSE*
- *STOI* מעל 70 אחוז – מעיד על רמת המובנות של הסיגנל ביחס ל- *reference*, ככל שהמספר גדל כך הוא יותר מובן.
- השיטה:** חישוב היחס בין ה- *Standard deviation* של החלק המופרד שנוצר מהפלט של ה- *separator* לבין הרעש הוא ה- *SNR* של התוצר.

$$SNR = \frac{std_{ref}}{std_{noise}}$$

$$SNR_{dB} = 20 \log_{10}(SNR)$$

עבור שירים עם *Reference* נבצע חישוב *MSE* בין השערוכים לבין ה- *Reference* המתאים לו על פני כל הדגימות:  $MSE = \frac{1}{n} \cdot \sum_n (Original_n - Estimate_n)^2$  (1).

לחישוב *STOI* נשתמש בספריית *Pystoi* כדי לקבל את תוצאת החישוב בין כל שיר מופרד למקור.

2. **היעד:** המקורות המשוערכים ב- *output* של המערכת יהיו איכותיים למען המשתמש.

**המדד:** לביצוע מדידה על איכות התוצר נשתמש בשיטת *MOS (Mean Opinion Score)*, שיטה זו משתמשת בדירוגים כמותיים שהמשתמש נותן על איכות ההפרדה של התוצר והתוצאה הממוצעת על פני כלל הציונים שהתקבלו עבור כל מקור תייצג את האיכות שלו.

$$(2) \quad MOS = \frac{\sum_i X_i}{n} ; X_i = \text{Individual score} ; N = \text{Number of participants}$$

**שיטת ביצוע:** עריכת סקר הכולל השמעת שירים שהופרדו באמצעות המערכת למספר משתתפים שישמעו אותם ויתנו ציון לפי דעתם על כל מקור של שיר בנפרד.

## סקירת ספרות:

מודלים רבים להפרדת מקורות פועלים על רכיב העוצמה של האות, בדרך כלל זה גורר איבוד מידע של רכיב הפאזה שמקשה על ביצוע הפרדות איכותיות. המאמר מציג פתרון לביצוע הפרדת מקורות (End-to-End) תוך שמירה על רכיב הפאזה באמצעות שימוש ב-*Wave-U-Net*, שהיא רשת *CNN* להפרדת מקורות המתייחדת בכך שהיא פועל על האות בציר הזמן ולא דורשת התמרות לציר הספקטרום. על מנת להפיק תוצאות ראשוניות מהמודל אימנו אותו תחת *MUSDB18 Dataset* המכיל שירים עם ההפרדות שלהם לביצוע אימון מונחה וערכו השוואה בינם לבין תוצאות של מודל קיים, נמצא מתוצאותיהם שהמודל המוצע בעל ביצועים יותר טובים מה-*State-of-the-art* הנוכחי [1].

בחירת ה-*GPU* המתאים לאימון מערכות *Deep Learning* הוא קריטי מאחר ומערכות אלה דורשות כמות חישובים עצומה מאוד כך ש-*CPU* כבר אינו מספיק. *Tim Dettmers* הוא מתמחה לדוקטורט באוניברסיטת וושינגטון שעובד בתחום המחקר של ה-*Deep Learning* וביצוע אופטימיזציה של חומרה למול תוכנה. בסיקור שלו הוא מציג את היתרונות של ה-*GPU* כרכיב חומרה חשוב ואינטגרלי במערכות *Deep Learning*, הוא עורך סקירה של מספר דגמים שונים ומשווה ביניהם כעלות למול תועלת. החלופות החומרתיות למעבד שנבחר עבור פרויקט זה הסתמך על תוצאות הסיקור וכך נבחרה החלופה המרכזית [2].

מאמר נוסף המיישם את ה-*Wave-U-Net* להפרדת הזמר מהשיר לפי המודל המוצא ב-[1] ומשווה את המודל המאומן על פני *VCTK Dataset* למודל *State-of-the-art* באותם תנאים [3].

כחלק מפערי הידע לפרויקט יש לדעת כיצד עובדת ה-*Convolutional Layer* שהיא השכבה הבסיסית ביישום רשת ה-*Wave-U-Net* להפרדת מקורות. המאמר מציג את האג'נדה מאחורי תחום ה-*CNN* באינטליגנציה המלאכותית ומספק הסבר מפורט לכיצד עובדות שכבות אלו ברמת האלגוריתם ושימוש ב-*Feature maps* לביצוע פרדיקציות בהתאם ל-*Dataset* שעליו התקיים תהליך האימון למשימות כמו זיהוי עצמים, שחזור אודיו, עיבוד תמונה ועוד [4].

תהליך האימון של מערכות *Deep Learning* הוא בעצם תהליך אופטימיזציה של המודל למול ה-*Dataset* שעליו הוא מבוצע כך שהמודל הסופי יהיה האופטימלי ביותר. המאמר מציג אלגוריתם חדש לאופטימיזציה של *Stochastic Gradients* מסדר ראשון - *ADAM*, אשר יעיל באופן ביצוע החישובים וקל ליישום. המאמר מציג את המתמטיקה מאחורי האלגוריתם וכיצד הוא עובד, *pseudo* קוד ואנליזת התכנסות של האלגוריתם [5].

## ניתוח חלופות

### חלופה מערכתית:

פרויקט זה מציע ערכת קוד על גבי PC עם GPU שלאחר תהליך אימון תבצע הפרדת שירים למקורות באופן אוטומטי ומהיר ה- *Wave – U – Net*, כיום בשוק קיימות מערכות AI דומות וכמו כן גם תוכנות ייעודיות המבצעות את הפרדת השיר למקורות כמו *Audio DeMix*, *XTRAX STEMS* ו- *FASST*.

*Audio DeMix* בדומה ל- *XTRAX STEMS* הן תוכנות לעריכת אודיו שבאמצעותן ניתן לבצע הפרדות של שירים, הן עושות זאת באמצעות שרתים חיצוניים שעובדים עם GPU שמבצע את ההפרדה ולאחר מכן יורד למחשב דרך *Wi – Fi*. *FASST (Flexible Audio Source Separation Toolbox)* הוא כלי להפרדת שירים שזמין להורדה ושימוש באינטרנט.

נסקור את החלופות השונות למימוש הפרויקט וננתח את הבחירה בחלופת הפרויקט.

מדד / חלופה	<i>Wave – U – Net</i>	<i>Audio DeMix</i>	<i>XTRAX STEMS</i>	<i>Fasst</i>	משקל
מחיר	דורשת קניה חד פעמית של מחשב עם GPU 4	דורשת מנוי חודשי/שנתי לשימוש בתוכנה 1	דורשת מנוי חודשי/שנתי לשימוש בתוכנה 1	דורשת קניה חד פעמית של מחשב עם GPU 4	0.4
מהירות הפרדה	לאחר אימון המודל – ביצוע הפרדה בכמה שניות 5	שימוש בשרתים חיצוניים להפרדת השיר והורדת התוצרים למחשב לאחר הפרדתם - יכול לקחת זמן רב. 3	שימוש בשרתים חיצוניים להפרדת השיר והורדת התוצרים למחשב לאחר הפרדתם - יכול לקחת זמן רב. 3	ביצוע הפרדה בכמה שניות 5	0.2
איכות	מצומצם ל- <i>Dataset</i> עליו מתבצע תהליך האימון 3	עם נסיון בתוכנה המשתמש יכול לבצע הפרדות איכותיות 5	עם נסיון בתוכנה המשתמש יכול לבצע הפרדות איכותיות 5	מצומצם למודלים המוכנים מראש 1	0.3

אופן שימוש	הפעלת תהליך האימון מעט מסורבל אחת להפרדה נדרש רק שורת קוד אחת.	נדרשת למידת התוכנה	נדרשת למידת התוכנה	למידת ה- <i>ToolBox</i> ואופן שימוש במודלים	0.1
	3	4	4	4	
ציון משוקלל	3.8	2.9	2.9	3.3	

טבלה 1: ניתוח חלופות מערכתיות

החלופה הנבחרת לפרויקט משקלול סופי היא ה- *Wave – U – Net*.

### חלופה טכנולוגית – *GPU*:

אלגוריתמי *Deep Learning* דורשים כמות עצומה של חישובים בתהליך האימון, על מנת לקצר זמן זה באופן משמעותי *CPU* אינו מספיק וצריך להשתמש גם ב- *GPU*, שיועד להתמודד עם העברת כמות גבוהה של מידע באופן מהיר.

מדד / חלופה	NVIDIA-RTX2060	NVIDIA RTX2070	NVIDIA GTX 1070TI	משקל
מחיר [ש"ח] (ציון)	~1700 (5)	~2000 (4)	~2700 (2)	0.3
RAM [GB] (ציון)	6 (2)	8 (4)	8 (4)	0.4
base clock [Mhz] (ציון)	1365 (3)	1410 (4)	1607 (5)	0.1



0.2	2432	2304	1920	Num of cores. (ציון)
	(5)	(4)	(3)	
	3.7	4	3.2	ציון משוקלל

טבלה 2: ניתוח חלופות טכנולוגיות

חלופה נבחרת: Nvidia RTX2070.

יתרונותיה של החלופה הנבחרת לעומת האחרות בעיקר מתבטאת ביחס עלות מול תועלת מיטבי עבור תכנון והרצת אלגוריתמי אימון למערכת הלמידה בעמוקה, 8GB של זכרון RAM מתאים מאוד למערכות למידה עמוקה מתחילות ובהתאם לגודל הזכרון כך יאפשר להכניס יותר מידע לאימון באיטרציה אחת, כך גם כמות הליבות הקיימות בכל GPU, ובהתאם למחיר של החלופה – Nvidia RTX2070 נבחרה.

## תכן מפורט:

התכן המפורט יכול את כל שלבי הפרויקט: בחירת רכיבי החומרה, גרסאות תוכנה לספריות Python מתאימות, תיאוריה בסיסית, תיאור האלגוריתם, יישומו בקוד ואופן השימוש בתוצר הסופי.

### דרישות בסיסיות ליישום הפרויקט:

להלן יפורטו הדרישות הבסיסיות עליהן מושתת הפרויקט כאשר הן מחולקות ל 2 מרכיבים עיקריים: חומרה ותוכנה.

דרישות אלו נועדו על מנת להריץ את ה source code עבור פרויקט זה.

#### 1. דרישות חומרה:

a. NVIDIA RTX-2070 GPU (מפרט בנספח א').

b. Intel CPU i5-9400 (מפרט בנספח א').

c. 16GB RAM.

d. 1TB HDD.

בחירת רכיבי החומרה לפרויקט הייתה בקפידה רבה מכיוון שצריך לקחת בחשבון כמות עצומה של חישובים מטריציוניים (מכפלות, קונבולוציות, סכימות וכו') כך שבמידה והחומרה אינה מספקת הרצת הקוד יכולה לקחת זמן רב מידי, מאידך ניתן לקצר זמן זה ע"י בחירה נכונה של חומרה.

#### 2. דרישות תוכנה :

a. Python – 3.6.8.

b. Python packages:

יש לוודא התקנה של החבילות קוד הנ"ל על מנת להריץ את הקוד.

i. Numpy - חבילת קוד לתכנות בשפת Python אשר נותן את היכולת לבצע

פעולות מתמטיות רב-מימדיות גדולות.

ii. Sacred - חבילת Python המשמשת ככלי לביצוע קונפיגורציה, ארגון והוצאות

לוגים מהקוד. שינוי הקונפיגורציה רלוונטי עבור פרוייקט זה.

iii. Tensorflow-gpu - חבילה זו נועדה על מנת לאפשר לקוד לרוץ על ה GPU

דרך Python.

iv. Librosa - חבילת Python לאנליזה של אודיו ומוזיקה.

v. Soundfile - חבילת Python לקריאה/כתיבה של קבצי אודיו.

vi. Lxml - חבילת Python שמאפשרת עבודה יעילה עם קבצי XML, HTML.

vii. Musdb - חבילת Python לתהליך הניתוח של 18musdb sigsep שהוא בעצם

ה-dataset של הפרויקט (הסבר על כך בהמשך).

viii. Museval - חבילת Python לביצוע הערכה על שיערוכי מקורות מופרדים.

ix. Google - קישוריות למנוע חיפוש Google.

x. Protocol buffers - בשביל Google.

c. CUDA 9 for NVIDIA GPU.

סביבת עבודה למעבר הגרפי מאת NVIDIA אשר מאפשרת לפתח ולהריץ תוכנות מחשב על כרטיס ה-GPU, מיועד בעיקר למשימות עיבוד מקבילי מסיבי.

d. PyCharm.

סביבת העבודה עליו קוד ה-Python בנוי.

ההתקנה של החבילות שונות בגרסאותיהם המתאימות נעשתה באמצעות *PyPa* שהוא כלי עזר המיועד לכך, אפשרות ה-*Pip – install* מאפשרת התקנה קלילה של החבילות עפ"י קובץ ה-*Requierments* המוכל בחבילת הקוד לפרויקט זה.

## רשת קונבולוציה:

רשת קונבולוציה (CNN – Convolutional Neural Network) הינה אלגוריתם של Deep Learning שפותח בשנת 2012 ע"י אלכס קריזבסקי (Alex Krizhevsky) אשר זכה בתחרות ה-ImageNet Large Scale Visual Recognition Challenge, תחרות פתוחה לקהל שבה מפתחים אלגוריתמים שמטרתם לבצע זיהוי עצמים (Image Recognition) בתמונות הלקוחות מ-dataset של ImageNet.

רשת קונבולוציה מוגדרת ככזאת במידה והיא מכילה שכבות קונבולוציה בין השכבות החבויות, ה-*Hidden Layers*, שכבות אלו הן אלו שמגלות את המאפיינים של המידע, הייחודיות שלה היא ביכולת לגלות ("ללמוד") מאפיינים עבור המידע שהיא מקבלת ובאמצעות מאפיינים אלו היא מבצעת פרדיקציות למידע חדש.

רשת זו מכילה 3 סוגים שונים של שכבות:

1. 1d-Convolution Layer.

2. Max Pooling Layer.

3. Fully Connected Layer (FC Layer).

## שכבת הקונבולוציה:

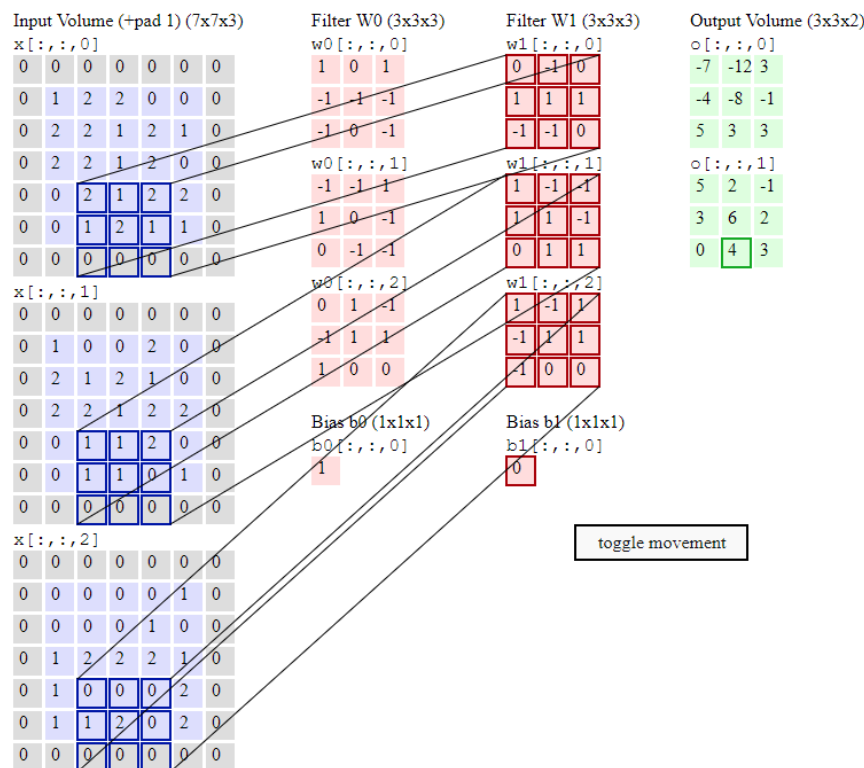
כשמה היא, מבצעים את פעולות הקונבולוציה בין המידע בכניסה לשכבה לבין מספר מסננים כאשר התוצר מכל פעולת קונבולוציה למסן אחד נקרא Feature map שזה אחד מן המאפיינים של אות המידע.

פעולת הקונבולוציה עובדת באופן הבא:

נתון שאות המידע בכניסה לשכבה מיוצג כמטריצה שגודלה  $N \times M$  וגודל המסננים הוא  $K \times K$ , מבצעים כפל איבר באיבר וסכימה (dot product) בין המסן לבין חתיכה מאות המידע כגודל המסן  $K \times K$  והמספר שיוצא נכנס למטריצת המוצא, מתחילים בראשית מטריצת אות המידע ומתקדמים בצעד אחד ימינה לכל אורך השורה עד שהיא נגמרת ולאחר מכן מתקדמים בצעד אחד למטה וחוזרים על השורה הבאה עד שעוברים על כל מטריצת הכניסה, ניתן גם לעשות יותר מצעד אחד והשינוי נקרא stride.

בסוף, נקבל מטריצה בגודל  $(N - K + 1) \times (M - K + 1)$  שהיא ה feature map שנוצרה עבור המסן, מכיוון שקיימים מספר מסננים של מסננים נוצרים בהתאם מספר זה של feature maps, כולן מאפיינים שחולצו מאות המידע ומשמשים כ- output לשכבה הבאה ברשת וכל שברשת יש יותר שכבות קונבולוציה כך נקבלת מאפיינים ברזולוציה יותר גבוהה.

דוגמא לפעולת הקונבולוציה בין אות מידע ל-2 מסננים והתוצר - feature map:



איור 3: אופן פעולת ה- 2d convolution

חשוב לציין שבפרויקט שלנו יש רק מטריצת Input Volume אחת (התמונה מתארת אות מידע של תמונה RGB ולכן 3 מטריצות) ולכן הקונבולוציה המתבצעת בשכבת הקונבולוציה הינה חד-מימדית.

### Pooling Layer:

בדרך כלל שכבה זו נמצאת מיד לאחר שכבת הקונבולוציה ומטרתה להקטין את גודל המימדים של מטריצת ה- feature map במוצא השכבה הקודמת, זאת על מנת להקטין את כוח החישוב הנדרש לעיבוד המידע ובנוסף המאפיינים המחולצים בשכבת הקונבולוציה הבאה יהיו מאפיינים יותר דומיננטיים [4].

### אופטימיזציה:

תהליך האופטימיזציה (תהליך האימון) ברשתות הינו תהליך איטרטיבי שבו בכל איטרציה מתבצע עדכון של משקלי ופרמטרי המודל על מנת לקבל מודל יותר טוב. העדכון של הפרמטרים מתבצע האמצעות מציאת ה- Gradient של ה- Loss function ולקיחת צעד מאוד קטן בכיוון המינימום של פונקציה זו – נקרא גם *Gradient Descent*.

$$(3) \quad w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^n \nabla_w Q(z_i, w_t),$$

כאשר:

$W_t$  - וקטור המשקלים באיטרציה  $t$ .

$\gamma$  – קצב הלמידה (Learning Rate).

$n$  – מספר הדגימות.

$Q(z, w)$  – Loss Function.

באמצעות שיטה זו בכל איטרציה נתקדם יותר ויותר לכיוון המינימום של ה- Loss function, משמע, התוצר הסופי לאחר מעבר בכל האיטרציות (ניתן להחליט המשתמש) יהיה המודל עם הפרמטרים הכי אופטימליים לבעיה אשר נותן את הפרדיקציות הטובות ביותר.

חשוב לציין שעל קצב הלמידה לא להיות נמוך מספיק שכך תהליך האימון יהיה ארוך מידי ולא יתכנס למינימום הרצוי (Under-Fitting) ושלא יהיה גבוה מידי שכך נפספס נקודות מינימום מקומיות על פונקציית ה- Loss (Over-Fitting).

עבור פרויקט זה נשתמש באופטימיזר ADAM – Adaptive Moment Estimation.

## :ADAM Optimizer

אופטימיזר ADAM הינו אלגוריתם לאופטימיזציית *Gradient* מסדר ראשון של פונקציות אקראיות [6].

האלגוריתם:

- מיושם בדרך ישירה (פונקציה מובנית של TensorFlow בלולאה).
- יעיל ברמת החישובים וצריכת זיכרון
- מתאים לרשתות בעלי מספר גדול של פרמטרים ומידע.

---

```
Require:  $\alpha$ : Stepsize  
Require:  $\beta_1, \beta_2 \in [0, 1)$ : Exponential decay rates for the moment estimates  
Require:  $f(\theta)$ : Stochastic objective function with parameters  $\theta$   
Require:  $\theta_0$ : Initial parameter vector  
 $m_0 \leftarrow 0$  (Initialize 1st moment vector)  
 $v_0 \leftarrow 0$  (Initialize 2nd moment vector)  
 $t \leftarrow 0$  (Initialize timestep)  
while  $\theta_t$  not converged do  
   $t \leftarrow t + 1$   
   $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )  
   $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)  
   $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)  
   $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)  
   $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)  
   $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)  
end while  
return  $\theta_t$  (Resulting parameters)
```

---

איור 4: psuedo code for ADAM

## :MUSDB18 Dataset

זהו ה-Dataset שבו נשתמש בפרויקט ועל בסיסו מתרחש אימון המודל, הוא מכיל 150 שירים מלאים מסגנונות שונים עם המקורות המופרדים מהם מראש, והוא חשוב לתהליך האימון כך שלמערכת יהיו "דוגמאות" ללמוד מהן כיצד לבצע את ההפרדה כדי להתאים אותה לצרכים שלנו.

ה-Dataset מכיל 2 תתי-תיקיות, אחת מהן נקראת Train המכילה 100 שירים לאימון המודל, השנייה נקראת Test ומכילה 50 שירים לבחינת המודל. בלמידה מונחית יש לבצע את אימון המערכת בשימוש של 2 תתי-התיקיות.

## Wave-U-Net:

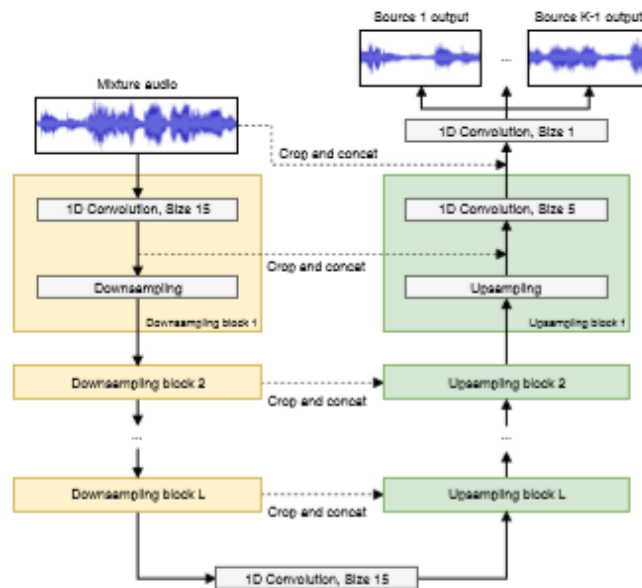
ה-Wave-U-Net הינה מודל לרשת קונבולוציה (CNN – Convolutional Neural Network), המיועד לפתירת משימות של הפרדת מקורות אודיו ופועל ישירות על מקור האודיו במרחב הזמן, מודל זה הוא הבסיס של הפרויקט מכיוון שהוא עונה על מטרתו המרכזית, להפריד שיר למספר מקורות.

המטרה שלנו היא להפריד מיקס  $M \in [-1,1]^{L_m \times C}$  ל-  $K$  מקורות  $S_1, \dots, S_K$ , כאשר  $S^k \in [-1,1]^{L_s \times C}$  לכל  $k \in \{1, \dots, K\}$ ,  $C$  הוא מספר הערוצים ו-  $L_m, L_s$  הם מספר דגימות האודיו של האות המקורי והמקורות המופרדים בהתאמה [1].

כאשר מעבירים את המידע מה- DataSet ברשת זו (מיקס כלשהו המורכב מארבעת המקורות), היא מחשבת מאפיינים עבור כל אחד מהמקורות ברזולוציות זמן שונות באמצעות בלוקי ה- Downsampling ו- Upsampling במשך  $L$  שכבות כאשר כל שכבה פועלת בחצי מרזולוציית הזמן מהקודמת לה בהתאמה ובאמצעות מאפיינים אלו היא מבצעת פרדיקציות מתאימות עבור שערכי המקורות  $K_i$  [1].

במוצא המודל אנו נקבל את שערכי המקורות  $S_k$  (Vocals, Bass, Drums, Others) עבור סט הפרמטרים הנוכחי, לאחר מעבר ב ADAM מתבצע עדכון לכל משקלי המודל כדי לשפר את ביצועי ההפרדה.

ארכיטקטורת המודל בנויה באופן הבא:



איור 5: ארכיטקטורה - Wave - U - Net

### **:Mixture Audio**

משמש כ-input למודל אליו נכנס השיר שאותו אנו רוצים להפריד, בתהליך האימון של המודל עוברים שירים מה-Dataset דרך בלוק זה ומהם מחולצים המאפיינים בשכבות הקונבולוציה שבבלוקים הבאים.

### **:Down-Sampling Block**

בלוק זה מורכב משכבת קונבולוציה המלווה בהפחתה של קצב הדגימה במשך  $L$  שכבות, באמצעות שיטה זו ניתן לחלץ מאפיינים בצירי זמן גסים יותר. קצב הדגימה המופחת בכל פעם מזניח כל מאפיין שני ממוצא שכבת הקונבולוציה כדי להקטין פי 2 את רזולוציית הזמן [1].

### **:Up-Sampling Block**

בלוק מורכב מהעלאת קצב הדגימה המלווה בשכבת קונבולוציה במשך  $L$  שכבות נוספות, לכניסה של שכבות הקונבולוציה בבלוקים אלו מצרפים את המאפיינים שחולצו בשכבת הקונבולוציה של בלוק ה-Down-Sampling המתאים לו לפי השכבה (Crop & Concat) כדי לקבל מאפיינים ברזולוציה גבוהה (High Resolution Features) [1].

### **:Difference Output Layer**

שכבה זו מחשבת את שערוכי המקור האחרון  $S^K$  ע"י הפחתת סך של השערוכים שחושבו מהמידע המקורי -  $S^K = M - \sum_{j=1}^{K-1} S^j$  (2) [2].

היתרון של שכבה זו הוא בהגדרה של  $M = \sum_{j=1}^K S^j$  אשר מונעת פלט לא סביר מהמודל, זאת עלולה להאט את תהליך האימון ולהפחתה בביצועים

## **המודל של הפרויקט ותהליך האימון:**

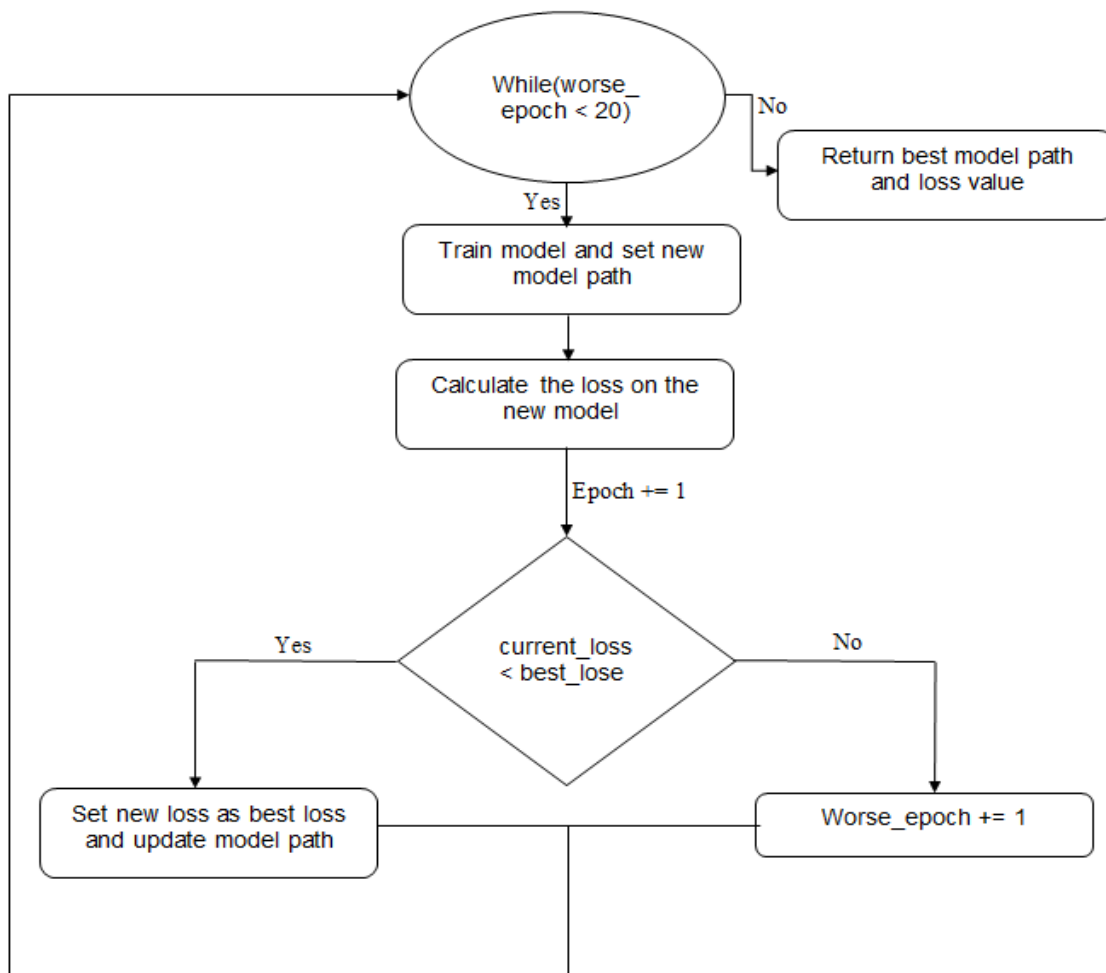
במהלך אימון המודל קבצי אודיו מה-*Train set* של ה-*Dataset* נדגמות רנדומלית בקבוצות למודל, לאחר ההפרדה המתבצעת ברשת נקבל שיערוכים של 4 מקורות (זמרת, בס, תופים, צלילים גובהים) לכל השירים בקבוצה. עליהם מתבצע חישוב MSE בין המקור המופרד המקורי ב-*Dataset* לבין השיערוך של המודל כחישוב ה-Loss הכולל של המודל.

עבור הפרויקט שלנו עפ"י [2] נבחר:



- $L_m = 147443, L_s = 16389$  [samples]
  - $L = 12$  layers
  - $F_c = 24$  פילטרים נופים עבור כל שכבת קונבולוציה ( $F_c^l = l * F_c$ ).
  - $f_d = 15, f_u = 5$  גדלי מטריצת הפילטרים עבור *DS Blocks, US Blocks* בהתאמה.
  - $Learning Rate = 0.0001$
  - $\beta_1 = 0.9, \beta_2 = 0.999$  פרמטרים ל-ADAM.
  - $Batch Size = 8$  גודלה של קבוצת השירים שנכנסת כל פעם ב-Input למודל, בהתאם ליכולות החומרה ניתן להגדיל מספר זה.
  - כל 2000 איטרציות של אימון נגדיר כ- *Epoch* ונדרוש עצירה של האימון לאחר 20 *Epochs* שאין שיפור בולידציה, כלומר, ה-Loss גדל מהמודל הקודם.
- בסיום כל האיטרציות נקבל את המודל טוב ביותר ובעל ה-Loss הנמוך ביותר, עליו מתבצע שלב נוסף של אימון הנקרא *Fine Tuning* עבורו:
- $Learning Rate = 0.0001 \rightarrow 0.00001$
  - $Batch Size = 8 \rightarrow 8 * 2$
- תהליך זה משפר עוד יותר את המודל מאחר והצעדים שעושים בכיוון המינימום אפילו יותר קטנים ומתכנסים כאשר מכניסים יותר מידע (שירים) בקבוצות למודל.

### דיאגרמת בלוקים של האלגוריתם למציאת המודל הטוב ביותר:



איור 6: דיאגרמת בלוקים לאלגוריתם למציאת המודל

- **While (worse\_epoch < 20)**: נבצע את כל תהליכי האימון ובדיקה בלולאה שבודקת האם חרגנו מהסף המינימלי של איטרציות גרועות אותו הגדרנו ל-20, במידה וחרגנו מסף זה מסתיים תהליך האימון והאלגוריתם יחזיר את הנתיב בו נשמר המודל הטוב ביותר וחישבו ההפסד שלו.

- **Train model and set new model path**: בבזק זה מתבצע תהליך האימון של המודל, השירים מה- Dataset ב- Training set עוזרים בקבוצות דרך ה- Wave-U-Net כדי לייצר את שיערוכי המקורות שלהם באופן הבא [2]:

Block	Operation	Shape
	Input	(16384, 1)
DS, repeated for $i = 1, \dots, L$	$\text{Conv1D}(F_c \cdot i, f_d)$ Decimate	(4, 288)
	$\text{Conv1D}(F_c \cdot (L + 1), f_d)$	(4, 312)
US, repeated for $i = L, \dots, 1$	Upsample Concat(DS block $i$ ) $\text{Conv1D}(F_c \cdot i, f_u)$	(16834, 24)
	Concat(Input)	(16834, 25)
	$\text{Conv1D}(K, 1)$	(16834, 2)

איור 7: דיאגרמת בלוקים מופשטת ל- Wave-U-Net

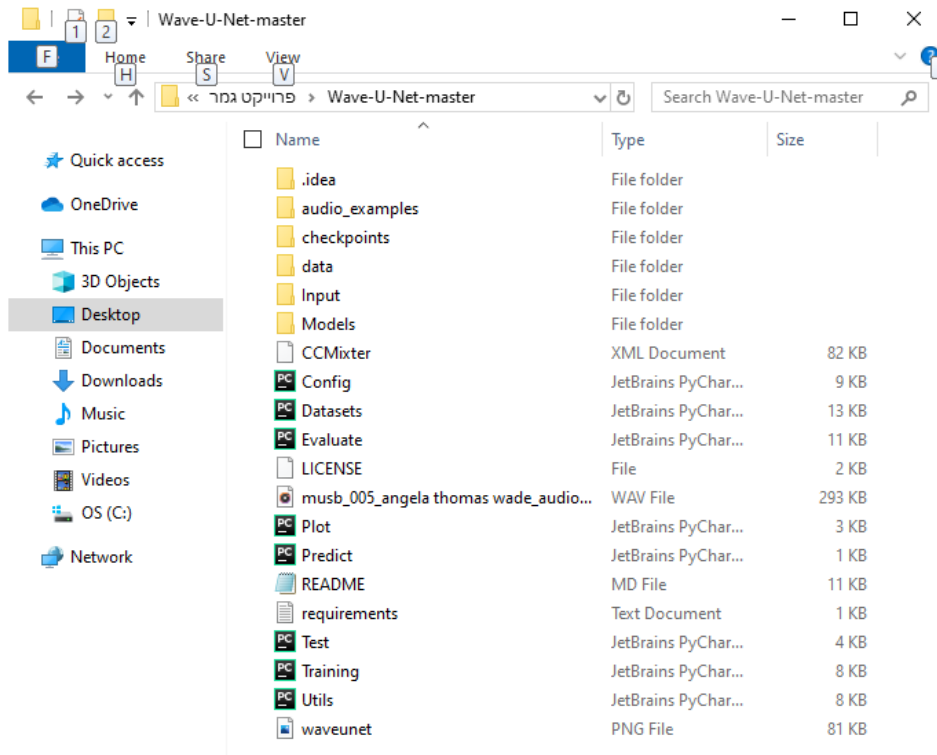
- לאחר מכן מתבצע חישוב של ה- Loss הכולל כפונקציית MSE על פני כל המקורות מופרדים ומתבצע עדכון למשקלי ופרמטרי המודל ע"י אופטימיזצור ADAM, במשך 2000 איטרציות. ניתן לראות את הקוד המיישם תהליך זה תחת נספח ב' – Training.py פונקציית Train().

- **Calculate the loss on the new model**: לאחר 2000 איטרציות של בלוק האימון מתבצע חישוב יותר גס של הפסד המודל מאחר ועורכים בדיקה בנוסף על שירים מה- Test set ב- Dataset שהם שירים שלא עברו בתהליך האימון של המודל, שירים אלו ושירים מה- Training set גם כן עוזרים במודל בקבוצות כדי לייצר את שיערוכי המקורות שלהם ועל פניהם מתבצע חישוב ה- Loss הכולל.

- **Current\_loss < Best\_loss**: מתבצעת בדיקה על ההפסד שחושב בבזק ה- Test למול ההפסד הטוב ביותר שקיבלנו עד כה בתהליך האימון. אם התנאי לא מתקיים נקדם ב-1 את פרמטר worse\_epoch, המונה איטרציות "גרועות", ונחזור שוב על תהליך האימון, אם התנאי מתקיים נאחזל את worse\_epoch חזרה ל-0 מאחר וקיבלנו שיפור ונבצע עדכון להפסד החדש ושמירת המודל לנתיב המודל הטוב ביותר.

## חבילת הקוד של הפרויקט:

פרויקט זה מיושם בקוד Python שהיא שפת Object-Oriented ברמה גבוהה, ספציפית בתחום ה deep learning כמות המידע ששפה זו מכילה גדולה מאוד ורוב הפיתוח מתבצע בה. חבילת



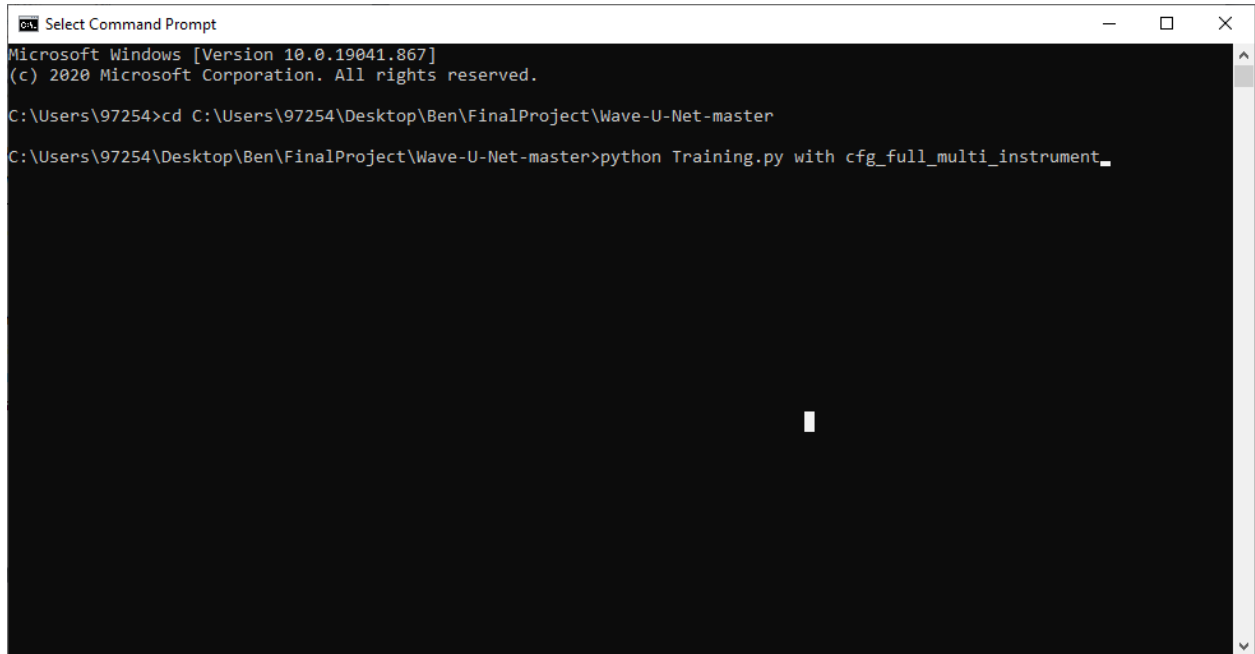
איור 8: חבילת הקוד

1. Config.py – קוד זה אחראי על עריכת קונפיגורציית המערכת, בתוכו הוא מגדיר מבנה בעל פרמטרים קבועים של המודל ומעדכן פרמטרים אחרים בהתאם למשימה. עפ"י הקונפיגורציה הנבחרת המבנה מתעכן ועובר לתוכנית המרכזית כפרמטר.
2. Datasets.py – קוד האחראי על תהליך ה- pre-preparing של ה- dataset. קיימות בו פונקציות עזר אשר תומכות בטעינת השירים ובהכנתם לכניסת רשת ה- Wave-U-Net בקוד.
3. Training.py – זהו הקוד המרכזי של הפרויקט בו מתבצעת מציאת המודל הטוב ביותר בעל ה-Loss הנמוך ביותר, הקוד זה מיושמת דיאגרמת הבלוקים שהוזכרה לעיל.
4. Utils.py – קוד המכיל פונקציות עזר בהן נשתמש במהלך הקוד על מנת להקל על סרבול הקוד ויישום של אותה משימה במקומות שונים בתוכנית.

5. Test.py – קוד זה מוכל ב-Traning.py ומיישם את בלוק ב- Calculate loss on the new model, הוא מחשב את ה-Loss של המודל עפ"י שירים ב- Test set של ה-Dataset בנוסף על שירים מה- Training set.
6. UnetAudioSeparator.py – קוד המיישם את ה-Wave-U-Net [1] ואחראי על תהליך בניית המודל ומפיק במוצאו את שיערוכי המקורות.
7. Evaluate.py - קוד זה אחראי על חישוב השיערוכים של שיר עבור מודל מסוים ובנוסף לחישוב השיערוכים של שירים מה-Dataset כחלק מתהליך האימון.
8. Predict.py – באמצעות קוד זה נבצע הפרדה של שירים באמצעות מודל נתון, לאחר מציאת המודל הטוב ביותר נשתמש במודל זה על predict.py על מנת להפיק 4 קבצים של המקורות המופרדים.
9. OutputLayer.py – קוד האחראי על יישום ה-Output Layer [1], שהוא חישוב שיערוך המקור ע"י החסרת סך המקורות המשוערכים מהמיקס השלם.
10. Checkpoints – התיקייה אליה נשמרים כל המודלים המיוצרים במהלך תהליך האימון.

## הרצת תהליך האימון ושימוש במודל:

ניתן להפעיל תוכניות *Python* דרך ה- *Command prompt* במחשב, נבחר את הנתיב בו נמצאת חבילת הפרוייקט וכדי להריץ את תהליך האימון למערכת נשתמש בפקודה `:python Training.py with cfg_full_multi_instrument.`



```
Select Command Prompt
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\97254>cd C:\Users\97254\Desktop\Ben\FinalProject\Wave-U-Net-master
C:\Users\97254\Desktop\Ben\FinalProject\Wave-U-Net-master>python Training.py with cfg_full_multi_instrument_
```

איור 9: הרצת התוכנית

הפקודה מריצה את התוכנית *Training.py* תחת ערכי הקונפיגורציה להפרדת מקורות ב *Config.py*, המתחילה את תהליך האימון של ה- *Wave – U – Net* למציאת המודל הטוב ביותר להפרדת מקורות עם ה- *Dataset* הנתון כפי שמתואר בדיאגרמת הבלוקים לעיל.

חשוב לציין, בהרצה הראשונית של התוכנית מתבצע גם תהליך של *Pre – preparation* של ה- *DataSet* כפי שמתואר ב-[1] ולא מתבצע עיבוד של השירים כלל בתהליך.

```

Starting!
Finished testing - Mean MSE: 0.0014494433326565483
Performance on validation set worsened to 0.0014494433326565483
EPOCH: 69
Dataset ready!
Training...
Sep_Vars: 10263498
Num of variables57
2020-09-24 09:06:06.012173: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1435] Adding vis
2020-09-24 09:06:06.016265: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:923] Device inte
2020-09-24 09:06:06.019923: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:929] 0
2020-09-24 09:06:06.022347: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:942] 0: N
2020-09-24 09:06:06.024880: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1053] Created Te
GeForce RTX 2070, pci bus id: 0000:01:00.0, compute capability: 7.5)
Num of variables171
INFO:tensorflow:Restoring parameters from checkpoints\678533\678533-138000
INFO - tensorflow - Restoring parameters from checkpoints\678533\678533-138000
Pre-trained model restored from file checkpoints\678533\678533-138000
2020-09-24 09:06:24.175546: I T:\src\github\tensorflow\tensorflow\core\kernels\data\shuffle_dataset_op.cc:94] Filling up
2020-09-24 09:06:34.566135: I T:\src\github\tensorflow\tensorflow\core\kernels\data\shuffle_dataset_op.cc:94] Filling up
2020-09-24 09:06:44.001998: I T:\src\github\tensorflow\tensorflow\core\kernels\data\shuffle_dataset_op.cc:94] Filling up
2020-09-24 09:06:54.002668: I T:\src\github\tensorflow\tensorflow\core\kernels\data\shuffle_dataset_op.cc:94] Filling up
2020-09-24 09:07:04.003064: I T:\src\github\tensorflow\tensorflow\core\kernels\data\shuffle_dataset_op.cc:94] Filling up
2020-09-24 09:07:14.176915: I T:\src\github\tensorflow\tensorflow\core\kernels\data\shuffle_dataset_op.cc:94] Filling up
2020-09-24 09:07:19.464678: I T:\src\github\tensorflow\tensorflow\core\kernels\data\shuffle_dataset_op.cc:129] Shuffle b
Finished epoch!
Dataset ready!
Testing...
2020-09-24 09:22:16.472610: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1435] Adding vis
2020-09-24 09:22:16.478068: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:923] Device inte
2020-09-24 09:22:16.482771: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:929] 0
2020-09-24 09:22:16.485809: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:942] 0: N
2020-09-24 09:22:16.516345: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1053] Created Te
GeForce RTX 2070, pci bus id: 0000:01:00.0, compute capability: 7.5)
Num of variables57
INFO:tensorflow:Restoring parameters from checkpoints\678533\678533-140000
INFO - tensorflow - Restoring parameters from checkpoints\678533\678533-140000
Pre-trained model restored for testing
Starting!
Finished testing - Mean MSE: 0.0014530964476157549
Performance on validation set worsened to 0.0014530964476157549
EPOCH: 70

```

איור 10: מחזור epoch שלם

בכל איטרציה בה נבנה מודל פרמטרי חדש להפרדת מקורות מתבצע תהליך ה *Training* ולאחר מכן תהליך ה *Testing*, כפי שהוסבר בתכן המפורט, שבסופו יוחלט האם האיטרציה הייתה טובה יותר וקיים שיפור בביצועי המודל הנוכחי במידה וקיים שיפור המודל נשמר בפרמטר המכיל את מיקומו במחשב.

בסוף התהליך מתבצעת אבולוציה של המודל הסופי על פני כל השירים מה- *Dataset* ובסיומה מודפסת הודעת סיום האימון ומשך זמן כולל.

```
99%|███████████| 148/150 [31:51:24<07:27, 223.50s/it]Testing...
2020-09-25 17:36:30.915223: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1435] Adding visible gpu devices: 0
2020-09-25 17:36:30.918698: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:923] Device interconnect StreamExecutor with
2020-09-25 17:36:30.922066: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:929] 0
2020-09-25 17:36:30.924496: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:942] 0: N
2020-09-25 17:36:30.927831: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1053] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0) with 14.0 GiB memory available
Geforce RTX 2070, pci bus id: 0000:01:00.0, compute capability: 7.5)
Num of variables:56
INFO:tensorflow:Restoring parameters from checkpoints\678533\678533-102000
INFO - tensorflow - Restoring parameters from checkpoints\678533\678533-102000
Pre-trained model restored for song prediction
vocals => SDR:-1.182dB, SIR:-8.217dB, ISR:4.138dB, SAR:0.551dB,
drums => SDR:6.770dB, SIR:8.453dB, ISR:11.804dB, SAR:7.835dB,
bass => SDR:-1.572dB, SIR:-4.494dB, ISR:6.672dB, SAR:2.193dB,
other => SDR:4.039dB, SIR:7.921dB, ISR:6.344dB, SAR:5.936dB,

99%|███████████| 149/150 [31:54:59<03:41, 221.11s/it]Testing...
2020-09-25 17:40:06.370480: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1435] Adding visible gpu devices: 0
2020-09-25 17:40:06.373474: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:923] Device interconnect StreamExecutor with
2020-09-25 17:40:06.377111: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:929] 0
2020-09-25 17:40:06.379582: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:942] 0: N
2020-09-25 17:40:06.382622: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1053] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0) with 14.0 GiB memory available
Geforce RTX 2070, pci bus id: 0000:01:00.0, compute capability: 7.5)
Num of variables:56
INFO:tensorflow:Restoring parameters from checkpoints\678533\678533-102000
INFO - tensorflow - Restoring parameters from checkpoints\678533\678533-102000
Pre-trained model restored for song prediction
vocals => SDR:-2.612dB, SIR:-1.126dB, ISR:7.633dB, SAR:4.773dB,
drums => SDR:6.036dB, SIR:11.264dB, ISR:9.833dB, SAR:6.510dB,
bass => SDR:1.116dB, SIR:0.497dB, ISR:14.893dB, SAR:1.735dB,
other => SDR:3.779dB, SIR:5.782dB, ISR:6.589dB, SAR:4.887dB,

100%|███████████| 150/150 [31:58:58<00:00, 767.59s/it]
INFO - Waveunet Training - Completed after 1 day, 23:21:52

C:\Users\97254\Desktop\Ben\FinalProject\Github\Wave-U-Net-master>
```

איור 11: בחינת אבולוציה סופית למודלים

כדי להשתמש במודל הסופי לאחר תהליך האימון נשתמש בפקודה `python Predict.py with cfg.full multi instrument`, הפקודה צריכה בנוסף לקבל את הנתיב למודל שבה נשתמש להפרדה `model_path`, ואת הנתיב לשיר שאותו רוצים להפריד `input_path`, במידה ולא נציין את הנתיב לתוצרים הם ייווצרו תחת ה- `input_path` כברירת מחדל.

```

C:\Users\97254\Desktop\Ben\FinalProject\Github\Wave-U-Net-master>python Predict.py with cfg.full_multi_instrument model_path="checkpoints\949470\949470-192000" input_path="D:\Ben\FinalProject\TestSongs_Train\Young Griffo - Pennies.stem.mp4
Training multi-instrument separation with best model
WARNING - Waveunet Prediction - No observers have been added to this run
INFO - Waveunet Prediction - Running command 'main'
INFO - Waveunet Prediction - Started
Producing source estimates for input mixture file D:\Ben\FinalProject\TestSongs_Train\Young Griffo - Pennies.stem.mp4
Testing...
2021-04-03 15:26:16.235439: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
2021-04-03 15:26:16.398680: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1356] Found device 0 with properties:
name: GeForce RTX 2070 major: 7 minor: 5 memoryClockRate(GHz): 1.62
pciBusID: 0000:01:00.0
totalMemory: 8.06GiB freeMemory: 7.01GiB
2021-04-03 15:26:16.398852: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1435] Adding visible gpu devices: 0
2021-04-03 15:26:16.843986: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:923] Device interconnect StreamExecutor with strength 1 edge matrix:
2021-04-03 15:26:16.844098: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:929] 0
2021-04-03 15:26:16.844567: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:942] 0: N
2021-04-03 15:26:16.845552: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1053] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 6768 MB memory) -> physical GPU (device: 0, name: GeForce RTX 2070, pci bus id: 0000:01:00.0, compute capability: 7.5)
Num of variables: 56
INFO: tensorflow: Restoring parameters from checkpoints\949470\949470-192000
INFO - tensorflow - Restoring parameters from checkpoints\949470\949470-192000
Pre-trained model restored for song prediction
INFO - Waveunet Prediction - Completed after 0:01:02

C:\Users\97254\Desktop\Ben\FinalProject\Github\Wave-U-Net-master>

```

אזור 12: ביצוע הפרדה ב-CMD



## התוצר:

התוצר הסופי של פרויקט זה הינו מודל מאומן להפרדת שיר למקורות מסוג *multi instrument*, המודל שאומן מכיל את הפרמטרים האופטמליים לביצוע הפרדות שירים בהתאם ל *Dataset* הנתון ויפריד כל שיר שיקבל ל-4 מקורות שונים תוך זמן קצר מאוד ובאמצעות פקודה אחת בלבד.

בנוסף, ערכת הקוד של הפרויקט מכילה את כל הקבצים ההכרחיים לביצוע אימון למודלים חדשים במידה ומתקיים שינוי ב- *Dataset* כמו החלפת סוגי השירים, הוספה של שירים וכו', כך שניתן ליצור מודלים נוספים שמותאמים לסוגי שירים שונים.

כל המודלים הנוצרים בתהליך האימון ניתנים לשמירה ולשימוש חוזר, אינם חד-פעמיים ואין צורך לבצע תהליך אימון נוסף.

## בדיקות והערכה:

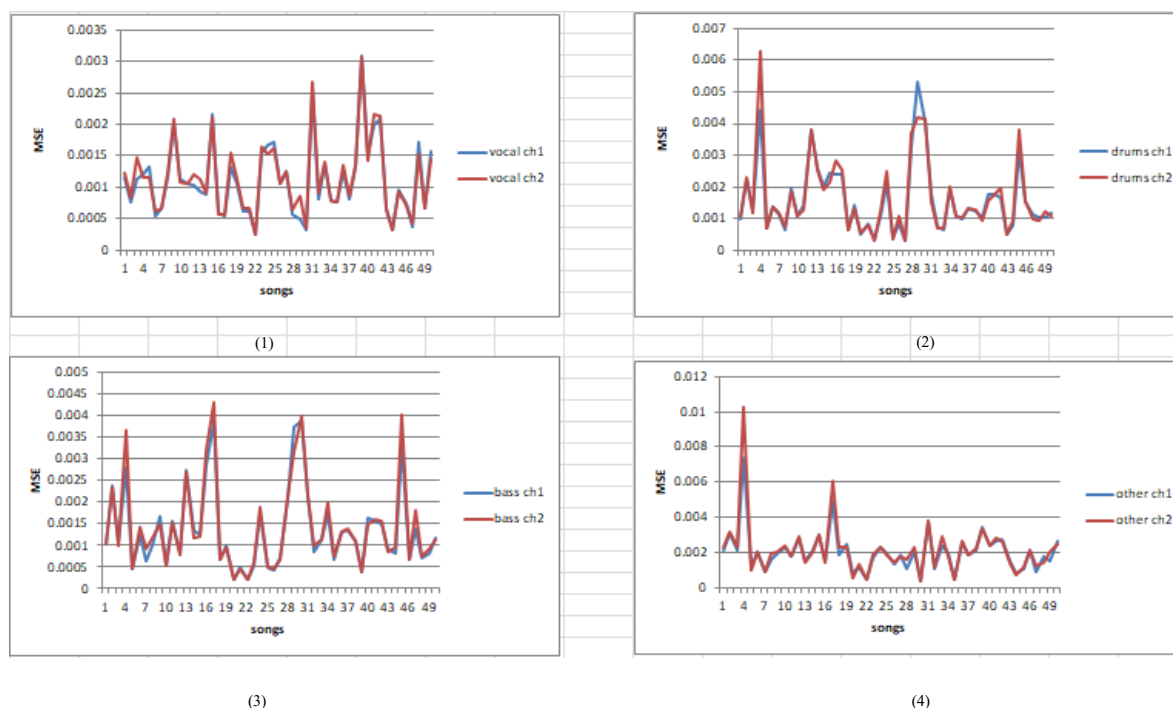
### **MSE:**

לאחר הרצת הקוד וקבלת המודל הטוב ביותר נשתמש בו כדי להפריד שירים מה-Dataset, מכיוון שקיימים ב-Dataset קבצים עם המקורות מופרדים מראש ניתן להשתמש בהם כ-Reference לשיערוכי המקורות שנקבל מהמודל ונבצע ביניהם חישוב MSE אופן הבא:

$$MSE = \frac{1}{n} \cdot \sum_n (original_n - estimate_n)^2$$

נדגום כל אחד מן השירים בקצב דגימה אחיד ונחסיר בין דגימות תואמות של המקור ושל ה-Reference, מכיוון שזה חישוב MSE נעלה את ההפרש בריבוע ונסכום את כל התוצאות ונחלק בסך כל הדגימות כדי לקבל את ממוצע של השגיאה.

בוצעו חישוב MSE למדגם של 50 שירים מתקייט ה-Test של ה-DataSet, התוצר הוא חישוב ב-2 ערוצים של mono ו-stereo ל-MSE, ככל שערך זה נמוך יותר כך קיים יותר דמיון בין השערוך למקור.



גרף 1-4: חישובי MSE לכל סיגנל בהפרדות שירים מה-Dataset

ניתן להסיק מתוצאות אלו שההפרדות שנעשו עבור השירים מה-Dataset היו קרובות מאוד למקוריות עבור כל אחד מן המקורות Vocals, drums, bass, others.

רוב התוצאות דומות עד כדי פקטור של  $10^{-3}$ , אמנם קיימים שירים מסויימים בהם ההפרדה פחות טובה עד כדי פאטקור של  $10^{-2}$ .

## :SNR

חישוב ה-SNR נועד לתאר את היחס בין הסיגנל הרצוי בשיר המופרד שלנו לעומת כמות הרעש בו. בדומה לחישוב ה-MSE, נבצע הפרדה לשירים מה-*DataSet* בתקיית ה-*Train* באמצעות המודל הסופי של הפרוייקט, לשירים אלה גם כן קיימים קבצים מופרדים ללא רעשים שהוקלטו בנפרד המשמשים כ-*Reference* ויעזרו לנו בחישוב ה-SNR.

כדי לבצע חישוב זה נשתמש בחישוב ה-*Standard deviation* של הרעש והחלק המופרד. בשביל הרעש – נדגום את השיר המופרד ואת ה-*Reference* ונחסר ביניהם, כך נקבל רק רעש.

$$Noise = Signal_{estimated} - Signal_{ref}$$

לאחר מכן נבצע חישוב *STD* עבור הרעש וה-*Reference* בנפרד באמצעות.

$$std_{Noise} = STD(Noise)$$

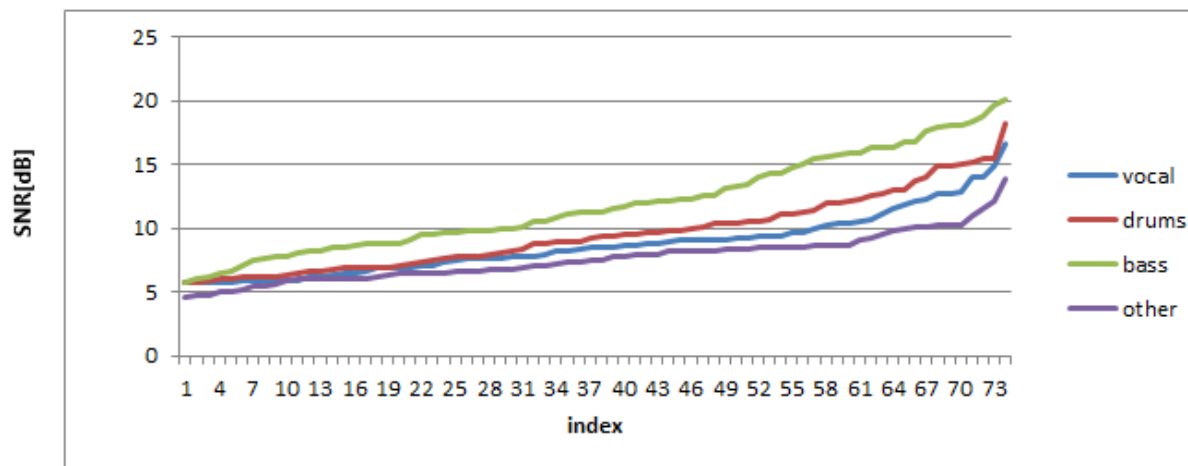
$$std_{ref} = STD(Signal_{ref})$$

נחלק בין החישוב של ה-*Reference* לחישוב של הרעש ונקבל את ה-SNR.

$$SNR = \frac{std_{ref}}{std_{noise}}$$

$$SNR_{dB} = 20 \log_{10}(SNR)$$

החישובים בוצעו על כל השירים בהם המודל נעזר בתהליך האימון שלו הנמצאים תחת תקיית *Train* של ה-*DataSet*



גרף 5: חישוב SNR להפרדות שירים מה-*Training set*.

## מהגרף ניתן להסיק:

- מרבית השירים הם בעלי  $SNR$  של יותר מ- $10dB$ , בשירים אלו ניתן לשמוע בבירור את החלק המופרד אך עדיין ניתן להבחין ברעש בחלקים מסויימים.
- בשירים בעלי  $SNR = 6[dB]$  ומטה הרעש יותר עוצמתי ויותר קשה להחבין בחלק המופרד.
- להפרדת ה- $Bass$  יש את התוצאות הכי טובות בהיבט  $SNR$  ומגיע למקסימום של  $20.17[dB]$ .
- להפרדת ה- $Other$  יש את התוצאות הפחות טובות מבין שאר ההפרדות.

## **MOS:**

לשם מציאת מדד  $MOS$  נערך סקר בין קבוצה בעלת 20 משתתפים פורמט לסקר בנספח ג'), המשתתפים התבקשו לשמוע 5 שירים ואת ההפרדות שלהם למשך כחצי דקה ולדרג את איכות ההפרדה לפי דעתם, השירים שנבחרו לסקר אינם חלק מה- $Dataset$  ולכל משתתף הושמעו אותם שירים לאחידות התוצאות.

לאחר שכלל הדירוגים התקבלנו נחשב את ממוצע הדירוגים של כל מקור בשיר עבור כלל המשתתפים, כך נקבל את הדירוג הממוצע עבור כל שיר למקור מסויים ונוכל לבצע חישוב נוסף לממוצע הדירוגים של 5 השירים כדי לקבל את הדירוג הסופי להפרדה של כל המקור בביצועי המודל.

לאחר קבלת הדירוגים מכלל המשתתפים התוצאות נסכמו באופן הבא:

$$Source_i = \sum_{k=0}^4 \frac{\sum_{j=0}^{19} Score_{kj}}{20} = \frac{1}{100} \sum_{k=0}^4 \sum_{j=0}^{19} Score_{ikj};$$

$i = Vocals, drums, bass, other; k = song number; j = participant$

תוצאות המדד מראות על תוצאות גבוהות של שכלול ממוצע הדירוגים, לפי החישוב  $Source_i[vocals, drums, bass, other] = [7.95, 7.9, 6.53, 8.07]$  ציון משוכלל למודל - 7.6125.

לפי חוות דעת של המשתתפים מעבר לרעשי הרקע בחלקים שקטים להפרדה כאשר היא קיימת מופרדת שומעים אותה באופן ברור לעומת שאר הכלים.

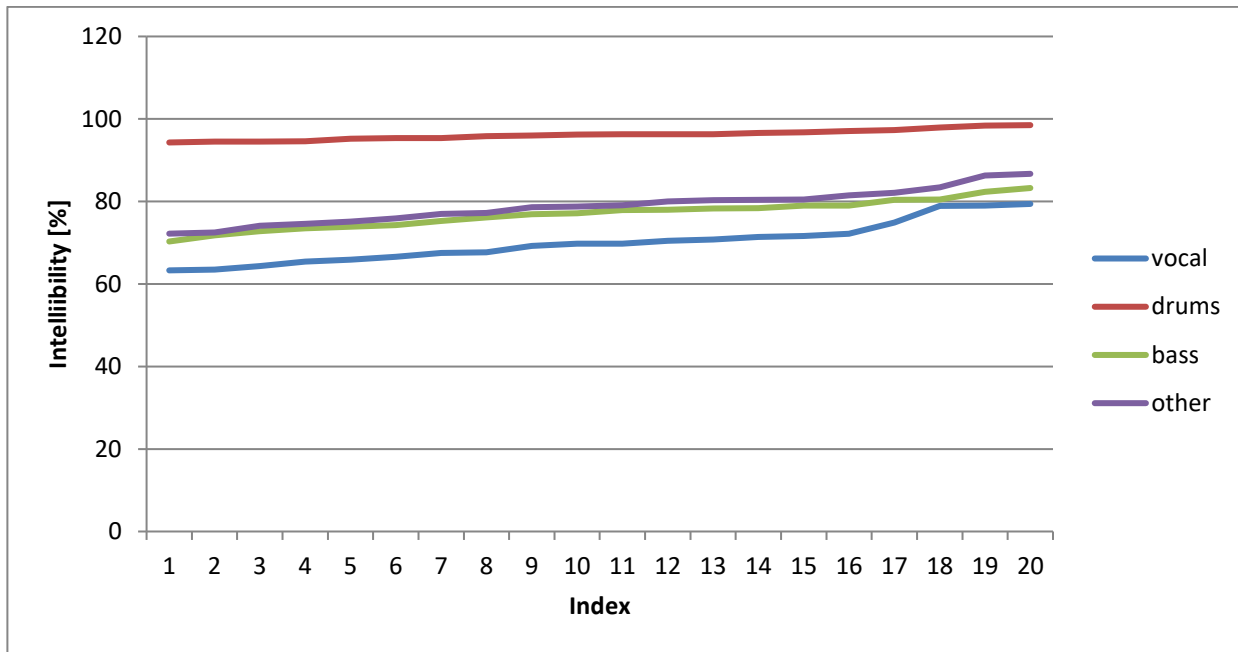
## **STOI**

מדד  $STOI$  הינו מדד איכות אשר נותן אינדיקציה מספרית באחוזים לרמת המובנות של שיר מסויים ביחס ל- $Reference$  קיים, ניתן לקבל תוצאה למדד זה באמצעות ספריית  $pystoi$  שבה קיימת הפונקציה לחישוב הפרמטר  $Stoi$ .

הפונקציה מקבלת בכניסתה את החלק המופרד בשיר שהתקבל בפלט מהמודל הסופי, את החלק המופרד בשיר המקורי מה- $Dataset$  וקצב הדגימה שבה נדגמו השירים ומחזירה ערך באחוזים כך שערך גבוה יותר מצביע על מובנות גבוה יותר של ההפרדה.

התוצאות של מדד זה עבור המודל הסופי חושבו על כל השירים מתקיימת  $Train$  של ה- $Dataset$  ולכל הפרדה בנפרד ( $vocals, drums, bass, other$ ), השירים נדגמו בקצב דגימה של  $fs = 16kHz$  בערוץ יחיד בכניסתם לפונקציה, עבור שירים ארוכים החישוב התבצע על פני קטע של 3 דקות.

נבחן את התוצאות על 20 שירים המובילים באחוזי התאמה למדד STOI:



גרף 1: תוצאות מדד STOI

מהגרף ניתן להסיק:

- כל ההפרדות של השירים מעל 60% התאמה במדד STOI.
- הפרדת ה- *drums* היא בעלת התוצאות הטובות ביותר ובה התוצאה הכי טובה למובנות ההפרדה ביחס למקור היא 98.5%.
- כל ההפרדות של *Other, Bass* גם כן מעל ל- 70% התאמה במדד, המראה על שהמדד אחיד בין כל ההפרדות.

## סיכום ומסקנות:

לסיכום, פרויקט זה עסק ביישום מערכת תוכנה מותקנת על PC המשתמשת באלגוריתמים של Deep Learning לפתרון משימת הפרדת שירים למקורות של זמר/ת, תופים, בס וצלילים אחרים. לשם כך השתמשנו ביישום ה-Wave – U – Net, שהוא אלגוריתם ללמידה עמוקה מסוג CNN המיועד להפרדת מקורות מקצה לקצה בציר הזמן.

תחילה ביצענו תהליך אימון מורכב בעל 2 חלקים (אימון ראשוני ו-Fine Tuning) שבו העברנו מידע מקוטלג (שירים וההפרדות שלהם) דרך הרשת שלפיו המערכת למדה להבחין בין הרכיבים השונים של השיר ולבצע הפרדות באמצעות בניית מודל פרמטרי המסתמך על נתונים שנלקחים משירים אלו ובסוף התהליך התקבל מודל סופי מבין רבים שנבחנו שהוא בעל הביצועים הטובים ביותר בהפרדות השירים.

באמצעות המודל הסופי שנוצר בוצעו הפרדות של שירים מה-Dataset ומיטויב בשביל מספר בדיקות שמטרתן לבחון את ביצועיו בקוד Python שצורף לקוד (נספח ב' -  $mse$ ):

- מדד MSE המתאר את הפרש השגיאה הממוצעת בין ההפרדה למקור נבחן על 50 שירים מסט ה-Test הראה על תוצאות טובות עם שגיאה ממוצעת של עד  $10^{-3}$ .
- מדד SNR לחישוב יחס אות לרעש נבחן על 75 שירים מסט ה-Train לא היה מדד אופטימלי לבדיקה אך הראה תוצאות טובות של 10dB ומעלה ברוב המקרים.
- מדד STOI המתאר את רמת המובנות של ההפרדה ביחד למקור נבחן על 20 שירים המובילים מסט ה-Train שיקפו ברובן תוצאות מעולות של מעל 70% התאמה.

מדד MOS המתאר חווית משתמש נבחן על 5 שירים חדשים שהופרדו ע"י המודל ודורגו ע"י 20 משתתפים שהעבירו את חוות דעתם על איכות ההפרדה עבור כל מקור מופרד, לפי המשתתפים מעבר לרעשי רקע ההפרדה נשמעת בבירור והציון הכללי למודל שהתקבל הוא 7.6125.

לפי תוצאות המדדים ניתן לראות שביצועי המודל שנוצר טובים מאוד בהיבטי איכות ומובנות על פני כל ארבעת ההפרדות של השירים (זמר/ת, בס, תופים, צלילים אחרים), סה"כ בוצעו כ-150 הפרדות של שירים באמצעות המודל שנבחן.

ניתן לקבל מודל טוב יותר בהתאם לחומרת המחשב ושינויי הקונפיגורציה של הקוד, בפרויקט זה נעשה שימוש בחומרה מינימלית לאימון המערכת לפי דרישות המערכת להרצת אלגוריתמי למידה עמוקה ובוצעו בעקבות כך שינויי קונפיגורציה, לכן אפשר להסיק שביצועי המודל הינם בהתאם.

## הצעות לעבודת המשך:

כדי להמשיך ולפתח את תוצר הפרויקט ולנסות לקבל מערכת שמפרידה שירים עם ביצועים גבוהים יותר ניתן לעשות 2 דברים:

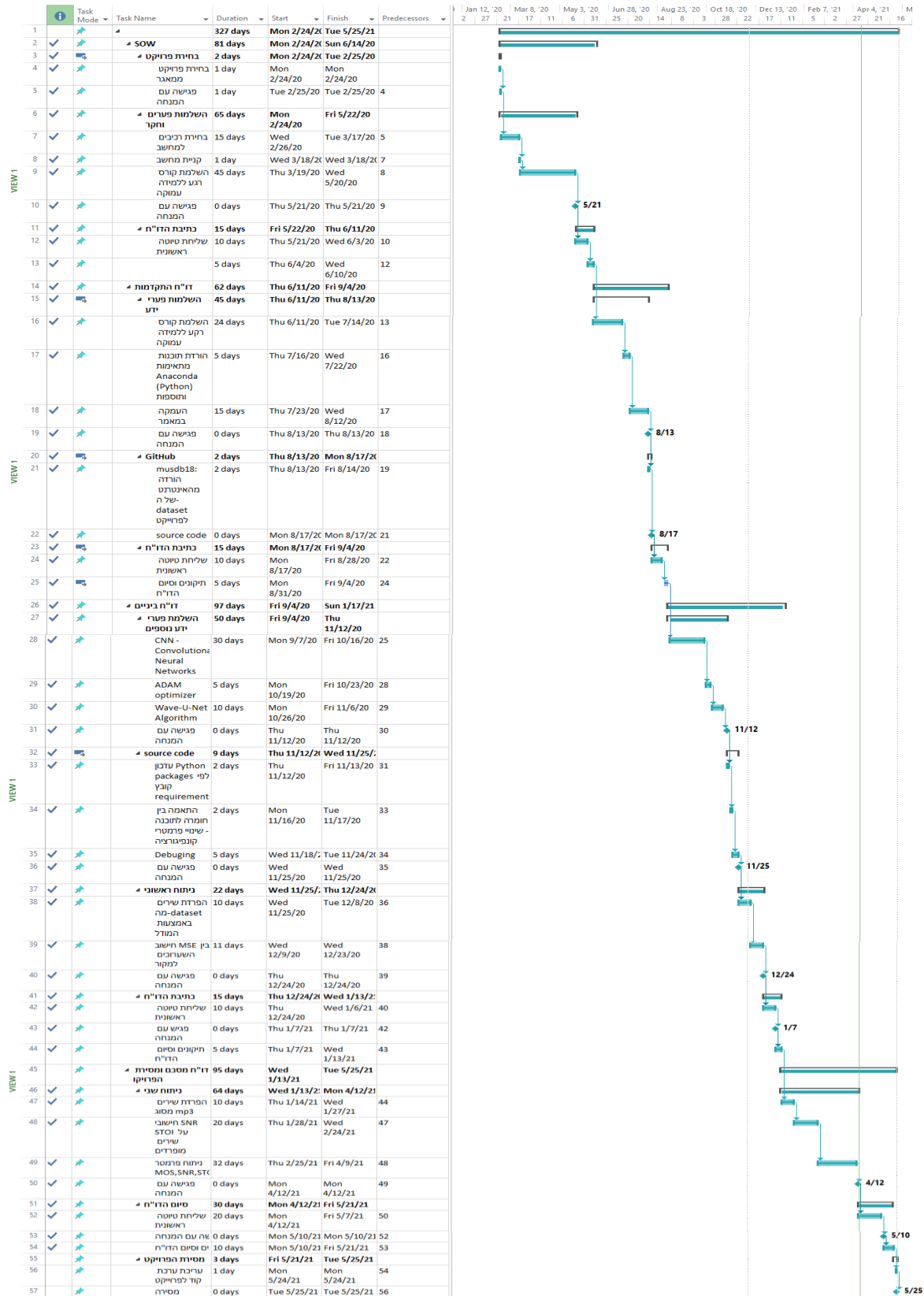
1. ניתן לבצע שינויים של חומרה ובהתאם לשנות את הקונפיגורציה. החלפה של רכיבי חומרה ברכיבים חזקים יותר אשר מגדילים את ביצועי המחשב מאפשרת לבצע שינויים של קונפיגורציית המערכת ב- *Config.py* וכך לשפר את זמן חישוב המודל הסופי בתהליך האימון ואת ביצועיו, הגדלת ה- *Cache\_Size* תאפשר ליותר מידע לעבור בו-זמנית במודל, הגדלת כמות הליבות שפועלות ב-*GPU* במידה וקיימות יותר באמצעות פרמטר *num\_workers*.

2. ניתן להרחיב את כמות השירים הנמצאים ה- *Dataset*, ככל שיש למערכת יותר מידע ללמוד ממנו בתהליך האימון שלה כך ביצועי המודל הסופי יגדלו בהתאם. תהליך האימון יהיה ארוך יותר לפי כמות השירים שנוספו ל- *Dataset* אך המודל הסופי כאמור יהיה בעל ביצועים טובים יותר.

עבור כל אחד מן השינויים שהוצעו יתקבל מודל שונה מן המודל הסופי שקיבלנו בפרויקט זה וניתן להשוות ביניהם באמצעות חישובי *STOI*, *MOS*, *MSE*, *SNR* ו- *STOI*.

# תכנון הפרויקט, ריכוז שינויים וניהול סיכונים:

א. תוכנית עבודה סופית – תרשים גאנט:





## ב. ריכוז שינויים:

- נוסף מדד מובנות ( $STOI$ ) לבחינת ביצוע הפרוייקט.
- עדכון קריטריון ליחס אות לרעש  $SNR[dB] = 10$ .

## ג. ניהול סיכונים:

- כמות המידע המועברת באלגוריתם גדולה מידיי לכדי שה-GPU יוכל להתמודד איתה ולכן ייקח זמן רב לאמן את המערכת, כדי להתמודד עם סיכון זה נבחר רכיב GPU מומלץ להרצת הקוד כך שהסיכון יורד משמעותית.
  - השלמת פערים – לימוד של חומר חדש באופן עצמאי והבנתו לעומק, ניתן סיוע מאנשים העוסקים בנושא ובנוסף קורס אינטרטי המלמד את הבסיס לתכנות Deep Learning ב-Python.
- במהלך הפרוייקט לא היו קשיים בסיכונים העיקריים שפורטו.

## רשימת מקורות:

- [1] Daniel Stoller, Sebastian Ewert, Simon Dixon, Wave-U-Net: a multi-scale neural network for end-to-end audio source separation, submitted at 2018-06-08.
- [2] Tim Dettmers, Which GPU(s) to get for deep learning: My experience and advice for using GPU's in deep learning, <https://timdettmers.com/2019/04/03/which-gpu-for-deep-learning/>, published at 2019-04-03.
- [3] Craig Macartny, Tillman Weyde, Improved Speech Enhancement with the Wave-U-Net, submitted at 2018-11-27.
- [4] Sumit Saha, A Comprehensive Guide to Concolutional Neural Networks – the ELI5 way, <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>, submitted at 2018-12-15.
- [5] Diederik P. Kingma, Jimmy Lei Ba, ADAM: a method for stochastic optimization, Published as a conference paper at ICLR 2015.

## נספח א' – מפרט מערכת CPU + GPU:

Processor Number <a href="#">?</a>	i5-9400F
Status	Launched
Launch Date <a href="#">?</a>	Q1'19
Lithography <a href="#">?</a>	14 nm
Use Conditions <a href="#">?</a>	PC/Client/Tablet
Recommended Customer Price <a href="#">?</a>	\$144.00 - \$157.00
<b>CPU Specifications</b>	
# of Cores <a href="#">?</a>	6
# of Threads <a href="#">?</a>	6
Processor Base Frequency <a href="#">?</a>	2.90 GHz
Max Turbo Frequency <a href="#">?</a>	4.10 GHz
Cache <a href="#">?</a>	9 MB Intel® Smart Cache
Bus Speed <a href="#">?</a>	8 GT/s
Intel® Turbo Boost Technology 2.0 Frequency* <a href="#">?</a>	4.10 GHz
TDP <a href="#">?</a>	65 W

## GEFORCE RTX 2070

### Memory Specs:

Memory Speed	14 Gbps	14 Gbps
Standard Memory Config	8 GB GDDR6	8 GB GDDR6
Memory Interface Width	256-bit	256-bit
Memory Bandwidth (GB/sec)	448 GB/s	448 GB/s

### GPU Engine Specs:

NVIDIA CUDA® Cores	2304	2304
RTX-OPS	45T	42T
Giga Rays/s	6	6
Boost Clock (MHz)	1710(OC)	1620
Base Clock (MHz)	1410	1410

```

import numpy as np
from sacred import Ingredient

config_ingredient = Ingredient("cfg")

@config_ingredient.config
def cfg():
    model_config = {"musdb_path" :
r"C:\Users\97254\Desktop\Ben\FinalProject\Github\train-test-dataset",
                    "estimates_path" :
r"C:\Users\97254\Desktop\Ben\FinalProject\Github\estimated_path",
                    "data_path" : "data",
                    "model_base_dir" : "checkpoints",
                    "log_dir" : "logs",
                    "batch_size" : 8,
                    "init_sup_sep_lr" : 1e-4,
                    "epoch_it" : 2000,
                    'cache_size': 4000,
                    'num_workers' : 4,
                    "num_snippets_per_track" : 100,
                    'num_layers' : 12,
                    'filter_size' : 15,
                    'merge_filter_size' : 5,
                    'input_filter_size' : 15,
                    'output_filter_size' : 1,
                    'num_initial_filters' : 24,
                    "num_frames": 16384,
                    'expected_sr': 22050,
                    'mono_downmix': True,
                    'output_type' : 'direct',
                    'output_activation' : 'tanh',
                    'context' : False,
                    'network' : 'unet',
                    'upsampling' : 'linear',
                    'task' : 'voice',
                    'augmentation' : True,
                    'raw_audio_loss' : True,
                    'worse_epochs' : 20,
                    }

    experiment_id = np.random.randint(0,1000000)
    # Set output sources
    if model_config["task"] == "multi_instrument":
        model_config["source_names"] = ["bass", "drums", "other", "vocals"]
    elif model_config["task"] == "voice":
        model_config["source_names"] = ["accompaniment", "vocals"]
    else:
        raise NotImplementedError
    model_config["num_sources"] = len(model_config["source_names"])
    model_config["num_channels"] = 1 if model_config["mono_downmix"] else 2

```

```
def full_multi_instrument():
    print("Training multi-instrument separation with best model")
    model_config = {
        "output_type": "difference",
        "context": True,
        "upsampling": "linear",
        "mono_downmix": False,
        "task" : "multi_instrument"
    }
```

## :Training.py

```
from sacred import Experiment
from Config import config_ingredient
import tensorflow as tf
import numpy as np
import os

import Datasets
import Utils
import Models.UnetSpectrogramSeparator
import Models.UnetAudioSeparator
import Test
import Evaluate

import functools
from tensorflow.contrib.signal.python.ops import window_ops

ex = Experiment('Waveunet Training', ingredients=[config_ingredient])

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)
@ex.config
# Executed for training, sets the seed value to the Sacred config so that
# Sacred fixes the Python and Numpy RNG to the same state everytime.
def set_seed():
    seed = 1337

@config_ingredient.capture
def train(model_config, experiment_id, load_model=None):
    # Determine input and output shapes
    disc_input_shape = [model_config["batch_size"],
model_config["num_frames"], 0] # Shape of input
    if model_config["network"] == "unet":
        separator_class =
Models.UnetAudioSeparator.UnetAudioSeparator(model_config)
    elif model_config["network"] == "unet_spectrogram":
        separator_class =
Models.UnetSpectrogramSeparator.UnetSpectrogramSeparator(model_config)
    else:
        raise NotImplementedError

    sep_input_shape, sep_output_shape =
separator_class.get_padding(np.array(disc_input_shape))
```

```

separator_func = separator_class.get_output

# Placeholders and input normalisation
dataset = Datasets.get_dataset(model_config, sep_input_shape,
sep_output_shape, partition="train")
iterator = dataset.make_one_shot_iterator()
batch = iterator.get_next()

print("Training...")

# BUILD MODELS
# Separator
separator_sources = separator_func(batch["mix"], True, not
model_config["raw_audio_loss"], reuse=False) # Sources are output in order
[acc, voice] for voice separation, [bass, drums, other, vocals] for multi-
instrument separation

# Supervised objective: MSE for raw audio, MAE for magnitude space
(Jansson U-Net)
separator_loss = 0
for key in model_config["source_names"]:
    real_source = batch[key]
    sep_source = separator_sources[key]

    if model_config["network"] == "UNET_spectrogram" and not
model_config["raw_audio_loss"]:
        window = functools.partial(window_ops.hann_window, periodic=True)
        stfts = tf.contrib.signal.stft(tf.squeeze(real_source, 2),
frame_length=1024, frame_step=768,
fft_length=1024, window_fn=window)
        real_mag = tf.abs(stfts)
        separator_loss += tf.reduce_mean(tf.abs(real_mag - sep_source))
    else:
        separator_loss += tf.reduce_mean(tf.square(real_source -
sep_source))
    separator_loss = separator_loss / float(model_config["num_sources"]) #
Normalise by number of sources

# TRAINING CONTROL VARIABLES
global_step = tf.get_variable('global_step', [],
initializer=tf.constant_initializer(0), trainable=False, dtype=tf.int64)
increment_global_step = tf.assign(global_step, global_step + 1)

# Set up optimizers
separator_vars = Utils.getTrainableVariables("separator")
print("Sep_Vars: " + str(Utils.getNumParams(separator_vars)))
print("Num of variables" + str(len(tf.global_variables())))

update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(update_ops):
    with tf.variable_scope("separator_solver"):
        separator_solver =
tf.train.AdamOptimizer(learning_rate=model_config["init_sup_sep_lr"]).minimiz
e(separator_loss, var_list=separator_vars)

# SUMMARIES
tf.summary.scalar("sep_loss", separator_loss, collections=["sup"])

```

```

sup_summaries = tf.summary.merge_all(key='sup')

# Start session and queue input threads
sess.run(tf.global_variables_initializer())
writer = tf.summary.FileWriter(model_config["log_dir"] + os.path.sep +
str(experiment_id),graph=sess.graph)

# CHECKPOINTING
# Load pretrained model to continue training, if we are supposed to
if load_model != None:
    restorer = tf.train.Saver(tf.global_variables(),
write_version=tf.train.SaverDef.V2)
    print("Num of variables" + str(len(tf.global_variables())))
    restorer.restore(sess, load_model)
    print('Pre-trained model restored from file ' + load_model)

    saver = tf.train.Saver(tf.global_variables(),
write_version=tf.train.SaverDef.V2)

# Start training loop
_global_step = sess.run(global_step)
_init_step = _global_step
for _ in range(model_config["epoch_it"]):
    # TRAIN SEPARATOR
    _, _sup_summaries = sess.run([separator_solver, sup_summaries])
    writer.add_summary(_sup_summaries, global_step=_global_step)

    # Increment step counter, check if maximum iterations per epoch is
    # achieved and stop in that case
    _global_step = sess.run(increment_global_step)

# Epoch finished - Save model
print("Finished epoch!")
save_path = saver.save(sess, model_config["model_base_dir"] + os.path.sep
+ str(experiment_id) + os.path.sep + str(experiment_id),
global_step=int(_global_step))

# Close session, clear computational graph
writer.flush()
writer.close()
sess.close()
tf.reset_default_graph()

return save_path

@config_ingredient.capture
def optimise(model_config, experiment_id):
    epoch = 0
    best_loss = 10000
    model_path = None
    best_model_path = None
    for i in range(2):
        worse_epochs = 0
        if i==1:
            print("Finished first round of training, now entering fine-tuning
stage")
            model_config["batch_size"] *= 2

```

```

        model_config["init_sup_sep_lr"] = 1e-5
        while worse_epochs < model_config["worse_epochs"]: # Early stopping
            on validation set after a few epochs
            print("EPOCH: " + str(epoch))
            model_path = train(load_model=model_path)
            curr_loss = Test.test(model_config,
model_folder=str(experiment_id), partition="valid", load_model=model_path)
            epoch += 1
            if curr_loss < best_loss:
                worse_epochs = 0
                print("Performance on validation set improved from " +
str(best_loss) + " to " + str(curr_loss))
                best_model_path = model_path
                best_loss = curr_loss
            else:
                worse_epochs += 1
                print("Performance on validation set worsened to " +
str(curr_loss))
            print("TRAINING FINISHED - TESTING WITH BEST MODEL " + best_model_path)
            test_loss = Test.test(model_config, model_folder=str(experiment_id),
partition="test", load_model=best_model_path)
            return best_model_path, test_loss

@ex.automain
def run(cfg):
    model_config = cfg["model_config"]
    print("SCRIPT START")
    # Create subfolders if they do not exist to save results
    for dir in [model_config["model_base_dir"], model_config["log_dir"]]:
        if not os.path.exists(dir):
            os.makedirs(dir)

    # Optimize in a supervised fashion until validation loss worsens
    sup_model_path, sup_loss = optimise()
    print("Supervised training finished! Saved model at " + sup_model_path +
". Performance: " + str(sup_loss))

    # Evaluate trained model on MUSDB
    Evaluate.produce_musdb_source_estimates(model_config, sup_model_path,
model_config["musdb_path"], model_config["estimates_path"])

```

## :Test.py

```

import tensorflow as tf
from tensorflow.contrib.signal.python.ops import window_ops
import numpy as np
import os

import Datasets
import Models.UnetSpectrogramSeparator
import Models.UnetAudioSeparator
import functools

def test(model_config, partition, model_folder, load_model):
    # Determine input and output shapes

```

```

    disc_input_shape = [model_config["batch_size"],
model_config["num_frames"], 0] # Shape of discriminator input
    if model_config["network"] == "unet":
        separator_class =
Models.UnetAudioSeparator.UnetAudioSeparator(model_config)
    elif model_config["network"] == "unet_spectrogram":
        separator_class =
Models.UnetSpectrogramSeparator.UnetSpectrogramSeparator(model_config)
    else:
        raise NotImplementedError

    sep_input_shape, sep_output_shape =
separator_class.get_padding(np.array(disc_input_shape))
    separator_func = separator_class.get_output

    # Creating the batch generators
    assert ((sep_input_shape[1] - sep_output_shape[1]) % 2 == 0)
    dataset = Datasets.get_dataset(model_config, sep_input_shape,
sep_output_shape, partition=partition)
    iterator = dataset.make_one_shot_iterator()
    batch = iterator.get_next()

    print("Testing...")

    # BUILD MODELS
    # Separator
    separator_sources = separator_func(batch["mix"], False, not
model_config["raw_audio_loss"], reuse=False) # Sources are output in order
[acc, voice] for voice separation, [bass, drums, other, vocals] for multi-
instrument separation

    global_step = tf.get_variable('global_step', [],
initializer=tf.constant_initializer(0), trainable=False, dtype=tf.int64)

    # Start session and queue input threads
    sess = tf.Session()
    sess.run(tf.global_variables_initializer())
    writer = tf.summary.FileWriter(model_config["log_dir"] + os.path.sep +
model_folder, graph=sess.graph)

    # CHECKPOINTING
    # Load pretrained model to test
    restorer = tf.train.Saver(tf.global_variables(),
write_version=tf.train.SaverDef.V2)
    print("Num of variables" + str(len(tf.global_variables())))
    restorer.restore(sess, load_model)
    print('Pre-trained model restored for testing')

    # Start training loop
    _global_step = sess.run(global_step)
    print("Starting!")

    total_loss = 0.0
    batch_num = 1

    # Supervised objective: MSE for raw audio, MAE for magnitude space
(Jansson U-Net)

```



```

separator_loss = 0
for key in model_config["source_names"]:
    real_source = batch[key]
    sep_source = separator_sources[key]

    if model_config["network"] == "unet_spectrogram" and not
model_config["raw_audio_loss"]:
        window = functools.partial(window_ops.hann_window, periodic=True)
        stfts = tf.contrib.signal.stft(tf.squeeze(real_source, 2),
frame_length=1024, frame_step=768,
fft_length=1024, window_fn=window)
        real_mag = tf.abs(stfts)
        separator_loss += tf.reduce_mean(tf.abs(real_mag - sep_source))
    else:
        separator_loss += tf.reduce_mean(tf.square(real_source -
sep_source))
    separator_loss = separator_loss / float(model_config["num_sources"]) #
Normalise by number of sources

    while True:
        try:
            curr_loss = sess.run(separator_loss)
            total_loss = total_loss + (1.0 / float(batch_num)) * (curr_loss -
total_loss)
            batch_num += 1
        except tf.errors.OutOfRangeError as e:
            break

    summary = tf.Summary(value=[tf.Summary.Value(tag="test_loss",
simple_value=total_loss)])
    writer.add_summary(summary, global_step=global_step)

    writer.flush()
    writer.close()

    print("Finished testing - Mean MSE: " + str(total_loss))

    # Close session, clear computational graph
    sess.close()
    tf.reset_default_graph()

    return total_loss

```

## :UnetAudioSeparator.py

```

import tensorflow as tf

import Models.InterpolationLayer
import Utils
from Utils import LeakyReLU
import numpy as np
import Models.OutputLayer

class UnetAudioSeparator:
    '''

```

```

    U-Net separator network for singing voice separation.
    Uses valid convolutions, so it predicts for the centre part of the input
    - only certain input and output shapes are therefore possible (see getpadding
    function)
    '''

    def __init__(self, model_config):
        '''
        Initialize U-net
        :param num_layers: Number of down- and upscaling layers in the
network
        '''
        self.num_layers = model_config["num_layers"]
        self.num_initial_filters = model_config["num_initial_filters"]
        self.filter_size = model_config["filter_size"]
        self.merge_filter_size = model_config["merge_filter_size"]
        self.input_filter_size = model_config["input_filter_size"]
        self.output_filter_size = model_config["output_filter_size"]
        self.upsampling = model_config["upsampling"]
        self.output_type = model_config["output_type"]
        self.context = model_config["context"]
        self.padding = "valid" if model_config["context"] else "same"
        self.source_names = model_config["source_names"]
        self.num_channels = 1 if model_config["mono_downmix"] else 2
        self.output_activation = model_config["output_activation"]

    def get_padding(self, shape):
        '''
        Calculates the required amounts of padding along each axis of the
input and output, so that the Unet works and has the given shape as output
shape
        :param shape: Desired output shape
        :return: Input_shape, output_shape, where each is a list [batch_size,
time_steps, channels]
        '''

        if self.context:
            # Check if desired shape is possible as output shape - go from
output shape towards lowest-res feature map
            rem = float(shape[1]) # Cut off batch size number and channel

            # Output filter size
            rem = rem - self.output_filter_size + 1

            # Upsampling blocks
            for i in range(self.num_layers):
                rem = rem + self.merge_filter_size - 1
                rem = (rem + 1.) / 2. # out = in + in - 1 <=> in = (out+1)/

            # Round resulting feature map dimensions up to nearest integer
            x = np.asarray(np.ceil(rem), dtype=np.int64)
            assert(x >= 2)

            # Compute input and output shapes based on lowest-res feature map
            output_shape = x
            input_shape = x

```

```

        # Extra conv
        input_shape = input_shape + self.filter_size - 1

        # Go from centre feature map through up- and downsampling blocks
        for i in range(self.num_layers):
            output_shape = 2*output_shape - 1 #Upsampling
            output_shape = output_shape - self.merge_filter_size + 1 #

Conv

            input_shape = 2*input_shape - 1 # Decimation
            if i < self.num_layers - 1:
                input_shape = input_shape + self.filter_size - 1 # Conv
            else:
                input_shape = input_shape + self.input_filter_size - 1

            # Output filters
            output_shape = output_shape - self.output_filter_size + 1

            input_shape = np.concatenate([[shape[0]], [input_shape],
[self.num_channels]])
            output_shape = np.concatenate([[shape[0]], [output_shape],
[self.num_channels]])

            return input_shape, output_shape
        else:
            return [shape[0], shape[1], self.num_channels], [shape[0],
shape[1], self.num_channels]

    def get_output(self, input, training, return_spectrogram=False,
reuse=True):
        """
        Creates symbolic computation graph of the U-Net for a given input
        batch
        :param input: Input batch of mixtures, 3D tensor [batch_size,
num_samples, num_channels]
        :param reuse: Whether to create new parameter variables or reuse
existing ones
        :return: U-Net output: List of source estimates. Each item is a 3D
tensor [batch_size, num_out_samples, num_channels]
        """
        with tf.variable_scope("separator", reuse=reuse):
            enc_outputs = list()
            current_layer = input

            # Down-convolution: Repeat strided conv
            for i in range(self.num_layers):
                current_layer = tf.layers.conv1d(current_layer,
self.num_initial_filters + (self.num_initial_filters * i), self.filter_size,
strides=1, activation=LeakyReLU, padding=self.padding) # out = in - filter +
1

                enc_outputs.append(current_layer)
                current_layer = current_layer[:, ::2, :] # Decimate by factor
of 2 # out = (in-1)/2 + 1

            current_layer = tf.layers.conv1d(current_layer,
self.num_initial_filters + (self.num_initial_filters *
self.num_layers), self.filter_size, activation=LeakyReLU, padding=self.padding)

```

```

# One more conv here since we need to compute features after last decimation

# Feature map here shall be X along one dimension

# Upconvolution
for i in range(self.num_layers):
    #UPSAMPLING
    current_layer = tf.expand_dims(current_layer, axis=1)
    if self.upsampling == 'learned':
        # Learned interpolation between two neighbouring time
        # positions by using a convolution filter of width 2, and inserting the
        # responses in the middle of the two respective inputs
        current_layer =
Models.InterpolationLayer.learned_interpolation_layer(current_layer,
self.padding, i)
    else:
        if self.context:
            current_layer =
tf.image.resize_bilinear(current_layer, [1,
current_layer.get_shape().as_list()[2] * 2 - 1], align_corners=True)
        else:
            current_layer =
tf.image.resize_bilinear(current_layer, [1,
current_layer.get_shape().as_list()[2]*2]) # out = in + in - 1
            current_layer = tf.squeeze(current_layer, axis=1)
            # UPSAMPLING FINISHED

            assert(enc_outputs[-i-1].get_shape().as_list()[1] ==
current_layer.get_shape().as_list()[1] or self.context) #No cropping should
be necessary unless we are using context
            current_layer = Utils.crop_and_concat(enc_outputs[-i-1],
current_layer, match_feature_dim=False)
            current_layer = tf.layers.conv1d(current_layer,
self.num_initial_filters + (self.num_initial_filters * (self.num_layers - i -
1)), self.merge_filter_size,
activation=LeakyReLU,
padding=self.padding) # out
= in - filter + 1

            current_layer = Utils.crop_and_concat(input, current_layer,
match_feature_dim=False)

# Output layer
# Determine output activation function
if self.output_activation == "tanh":
    out_activation = tf.tanh
elif self.output_activation == "linear":
    out_activation = lambda x: Utils.AudioClip(x, training)
else:
    raise NotImplementedError

if self.output_type == "direct":
    return Models.OutputLayer.independent_outputs(current_layer,
self.source_names, self.num_channels, self.output_filter_size, self.padding,
out_activation)
elif self.output_type == "difference":
    cropped_input =

```

```

Utils.crop(input,current_layer.get_shape().as_list(),
match_feature_dim=False)
        return Models.OutputLayer.difference_output(cropped_input,
current_layer, self.source_names, self.num_channels, self.output_filter_size,
self.padding, out_activation, training)
    else:
        raise NotImplementedError

```

**:mse**

```

import soundfile as sf
import numpy as np
from pystoi import stoi
from pesq import pesq
from scipy.io import wavfile
from scipy.signal import resample_poly
ref, fs_ref = sf.read('D:\Ben\FinalProject\Github\Train-test-
dataset\Train\Actions - South Of The Water.stem_vocals.wav') # input -
reference mixture path
pred, fs_pred = sf.read('D:\Ben\FinalProject\TestSongs_Train\Actions - South
Of The Water.stem.mp4_vocals.wav') # input - prediction from U-Net
if ref.ndim > 1:
    ref = np.mean(ref, axis=-1)

if pred.ndim > 1:
    pred = np.mean(pred, axis=-1)
if fs_ref > 16000:
    ref = resample_poly(ref[:4410000], 16000, fs_ref)
if fs_pred > 16000:
    pred = resample_poly(pred[:4410000], 16000, fs_pred)

print("pred sample:{}".format(fs_pred))
d = stoi(ref, pred, fs_ref, extended=False)
print("STOI:{}".format(d))
print(pesq(16000, ref, pred, 'wb'))
mse = np.sum((ref - pred)**2, axis=0) / len(pred) # calculation of MSE

noise = pred - ref
std_noise = np.std(noise)
std_ref = np.std(ref)
snr = std_ref / (std_noise + np.finfo(np.float32).eps)
snr_db = 20 * np.log10(snr + np.finfo(np.float32).eps)
print("SNR:{}".format(snr_db))
print("MSE = {}".format(mse))

```

## נספח ג' – פורמט סקר MOS:

### סקר איכות:

מטרת הסקר היא למצוא את מדד MOS (Mean Opinion Score) של כל מקור שמע של תוצר המערכת.  
בתקיית 'שירים מופרדים' ניתן לראות שעבור כל שיר קיימים 4 קבצים נוספים שהם מקורותיו המופרדים שהפיקה המערכת של פרויקט זה: זמר/ת, תופים, בס, צלילים אחרים.  
בטבלה הבאה עלייך לדרג בטווח של 1-10 (כאשר 1 – גרוע, 10 – מצוין) את איכות השמע של המקורות המופרדים לפי דעתך.

Song name / Source	Vocals	Drums	Bass	Others
<b>Doors Down - Here Without You 3</b>				
<b>Kings Of Leon - Use Somebody</b>				
<b>Green Day - American Idiot</b>				
<b>Green Day -Boulevard of Broken Dreams</b>				
<b>Life Is Beautiful (Acoustic)</b>				

כיצד האלגוריתם עובד?



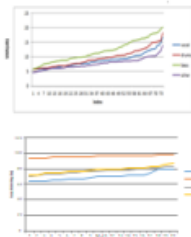
1. נרץ את תהליך האימון בלולאה במספר רב של איטרצציות כאשר התנאי הוא סף של מספר החזרות ה"גרועות".
2. בתהליך האימון מבצעים הפרדות של שירים מה-dataset שעבריים במודל על מנת ליצור את ה-feature maps המהווים את המשקלים של הרשת והבסיס לביצועי שיערוכים להפרדות.
3. נבצע אופטימיזציה לפרמטרי המודל באמצעות ADAM optimizer המעדכן את המשקלים בכיוון המינימום של ה-loss function שהוא פונקציה התלויה במשקלי המודל.
4. נבחנו את המודל הנוכחי ובחשואה למודל האחרון שנשמר ובמידה והוא טוב יותר נחליף בנרם, במידה ולא נקדם את מספר החזרות הגרועות.
5. בסוף התהליך מתקבל מודל סופי עם הפרמטרים הטובים ביותר לביצוע הפרדות שירים עליו מתבצעת אבולוציה סופית.



השתמשנו במודל הסופי שנוצר בתהליך האימון על מנת לבצע הפרדות של שירים מה-dataset ולבחון אותם בהשוואה לקבצי המקוד המופרדים.

סה"כ בוצעו הפרדות של כ-150 שירים מה-dataset באמצעות המודל לחישוב הפרמטרים.

לפי המדדים צפינו בתוצאות טובות של המודל המפרד בהיבט של איכות ומבונות הסיגנל הרצוי.

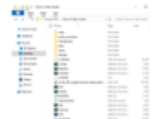


הפרדת זמר/ת וכלי נגינה מהקלטות של שירים

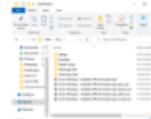
פרויקט זה מציג מערכת המבוססת על אלגוריתמי למידה עמוקה אשר תייצר מודל פרמטרי ממושקל להפרדת מקורות באמצעות תהליך אימון, המודל שנוצר יקבל בכניסתו קובץ אודיו המכיל שיר מלשהו ויפרידו באופן ברור ל-4 סיגנלים של זמר/ת, בס, תופים וצלילים אחרים.

דרישות עקרויות ליישום הפרויקט:

1. PC עם GPU.
2. תכנות בשפת PYTHON.
3. הכרת בסיסית של אלגוריתמי למידה עמוקה וסוגי שכבות נירונים.



ה-Wave-U-Net הוא אלגוריתם ללמידה עמוקה מסוג CNN המפיק מודל פרמטרי שבאמצעותו נפרד את השירים ופועל עליהם בציר הזמן וללא מעבר לציר התדר.



הוא מקבל בכניסתו שיר כלשהו ומחזיר במוצא 4 קבצים נפרדים עם המקורות המופרדים. קובץ בנפרד עבור הזמר/ת, קובץ בנפרד עבור הבס, קובץ בנפרד עבור התופים וקובץ בנפרד עבור צלילים אחרים.

ארכיטקטורת המודל של ה-Wave-U-Net:

1. אות המידע – שיר כלשהו.
2. מעבר של השיר ב-L שכבות קונבולוציה ודסימציה.
3. ביצוע קונבולוציה נוספת לרזולוציה הכי גבוהה.
4. מעבר ב-L שכבות אינטרפולציה וקונבולוציה נוספות.
5. ביצוע קונבולוציה נוספת להפקת K מקורות מופרדים.

