המחלקה להנדסת חשמל

שם הפרויקט: הפרדת כלי נגינה וזמר/ת מהקלטות של שירים.
Project Name: separation of musical instruments and singer recordings of songs.

דו"ח ביניים – midterm report

שם הסטודנט: בן ציון צוברי

מספר תעודת זהות: [redacted]

שם המנחה: שגיא הרפז

חתימת המנחה:

,

תוכן עניינים:

תקציר הדו"ח:

כיום עורכי אודיו משתמשים בתוכנות ייעודיות על מנת להפריד שירים בפורמטים שונים למקורות (זמר/ת, בד, תופים, צלילים אחרים), תוכנות אלו עולות כסף, לרוב לא נוחות לשימוש וצורכות זמן למידה של המשתמש, פרוייקט 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 האימון.
- קיימת אופציה ללמידה מונחית באמצעות Dataset.

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

ה- Wave-U-Net יודע לקחת שירים מ-Dataset קיים ולחלץ מהם "מאפיינים" שבאמצעותם הוא מחשב ב- output את שיערוכי המקורות של השיר ב- input . מאפיינים אלו נקראים ה- Feature Maps של המודל שנבנה ומכילים את הפרמרים של המודל שבאמצעותם הוא מבצע את השיערוך של המקורות.

ה- Feature Maps מחושבים בכל CL (Convolution Layer) ברשת באמצעות ביצוע 1dConvolution בין ה- input בכניסה לשכבה לבין מספר מסויים של פילטרים. Down/Up Sampling מבוצע בינCL כדי לקבל Feature Maps ברזולוציות זמן שונות. Feature Maps הן הבסיס של המודל בביצוע שערוכים לשירים.

לאחר שנקבל מודל מפריד עם Feature Maps ניתן לבצע לו תהליך Optimization שבו אנו נרצה לקבל את המודל עבורו ה- Loss Function מינימלית. תהליך זה נקרא גם תהליך האימון והרעיון שלו הוא להתקדם בצעדים קטנים לכיוון המינימום של ה- Loss Function.

בתהליך זה, שירים עוברים במודל במספר גדול של איטרציות, בכל אחת מהן נבנה מודל המכיל את הפרמטרים שלו ב- Feature Maps ואלו עוברים עדכון לכיוון המינימום באמצעות ADAM Optimizer עם מקדם LR (Learning Rate) נמוך כדי להמנע מ-Over/Under Fitting, עבור כל מודל בכל איטרציה מתבצע חישוב של ה- Loss הכולל שלו כחישוב MSE (Mean Squared Error) בין המקור לשערוכים ב- output כמדד לטיב המודל, במידה וחל שיפור שומרים אותו בימקום הנוכחי.

על מנת לשפר את המודל יותר נבצע תהליך אימון משני מייד בסיום הסבב הראשון הנקרא Fine Tuning שבו מקשיחים את התנאים לאימון כך שמבצעים צעדים אפילו קטנים יותר לכיוון המינימום של ה- Loss Function וממשיכים בביצוע סט איטרציות נוסף של אימון.

בסיום התהליך התקבל מודל הפרדה סופי בעל אחוז הולידציה הטוב ביותר על פני ה-Dataset שבאמצעותו כעת ניתן להפריד שירים ולערוך בדיקות וניסויים על תוצריו.

בניסויים ראשונים בוצעה הערכה לפרמטר MSE של המודל באמצעות ביצוע הפרדה לשירים מה- Dataset שבהם השתמשנו בתהליך האימון, לשערוכים של שירים אלו קיימים קבצי מקור (Reference) כך שניתן לחשב את השגיאה בין השערוך למקור. ככל ששגיאת ה- MSE נמוכה יותר כך ההפרדה יותר "דומה" למקור.

מתוצאות ראשונות קיבלנו ערכים נמוכים מאוד (בפאקטור 10^{-3}) של MSE על פני כל המקורות Vocals, drums, bass, others, עדיין קיימים שירים בודים בהם הערך יותר גבוה עבור מקורות בודדים.

בשלב הבא של הבדיקות נבצע חישוב MOS (Mean Opinion Score) המסתמך על חישוב של תוצאות אובייקטיביות של אנשים המדרגים את איכות ההפרדה בסקלה של מ 1-10 ובנוסף על חישוב SNR (Signal to Noise Ratio) עבור שירים מופרדים שאין להם קבצי מקור כעוד מדד לטיב ההפרדה.

האלגוריתם המוצע בפרוייקט זה מסתמך על המאמר ב- [1] שמציג את ה- Wave-U-Net כפתרון לבעיות של הפרדת מקורות ומציע שיטה ליישום האלגוריתם בפרמטרים ספציפיים של הרשת,

סקירת ספרות וסקר שוק:

מקורות חדשים:

מדריך מקיף לרשתות קונבולוציה (CNN), המאמר מתאר באופן כללי את אופן פעולת האלגוריתם ומפרט את כלל השכבות אשר מרכיבות רשת זו. הוא מדגיש את יתרונותיה של הרשת על פני אלגוריתמים אחרים ואת השימוש הנרחב שנעשה בה במסגרת תחום ה Computer Vision [5].

המאמר מציע 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 [6].

מטרות הפרויקט, יעדים ומדדים:

מטרה מרכזית:

מטרת פרויקט זה היא לתכנן מערכת אשר בכניסתה תקבל שיר, בתור קובץ מסוג, ותדע להפרידו ל-4 מקורות נפרדים – זמר/ת, תופים, בס, צלילים גבוהים.

יעדי הפרויקט:

1. **היעד:** יכולת הפרדה ברזולוציה גבוהה, על מנת להבחין באופן ברור שקיימת הפרדה בין השיר לסיגנל הרצוי.

המדד: נדרוש יחס אות לרעש (SNR) של לפחות 20[dB] על מנת לקבל את ההפרדה הרצויה על כל מקור משוערך (זמר/ת, בס, תופים, צלילים גבוהים).

כדי לחלץ את ה- SNR של השיר המופרד נשתמש בתיאור הספקטרלי שלו, בתיאור זה יהיו קיימים כל התדרים הרצויים לעומת התדרים הלא רצויים בעוצמות שונות ובאמצעות חישוב היחסים באופן הבא: $SNR = 20\log(\frac{P_{signal}}{P_{noise}})[dB]$ נוודא שהוא נמצא בתחום הרצוי לקבלת רזולוציה טובה.

עבור שירים עם Reference נבצע חישוב MSE בין המקור המשוערך לבין ה-Reference על פני כל הדגימות $MSE = \frac{1}{n} \cdot \sum_n (original_n - estimate_n)^2$.

2. **היעד:** הסיגנל ב- Output יהיה איכותי לאחר ההפרדה למען המשתמש.

המדד: על מנת לבצע מדד על איכות הסיגנל (מציאת מדד איכותי) נשתמש בשיטת MOS (Mean Opinion Score), אשר משתמשת בדירוגים של התוצאות השונות שיתקבלו בסקלה מסויימת ועל פי הדירוג הממוצע של כולם, התוצאה תצביע על איכות ההפרדה של האלגוריתם.

$$X_i = individual\ score\ ; n = num\ of\ participants\ ; MOS = \frac{\sum_i X_i}{n}$$

יש לבצע את הדירוג על כל מקור בנפרד, כלומר, יש לשכלל סה"כ 4 תוצרות MOS – עבור vocals, bass. Drums and others.

שיטת ביצוע: השמעת מיקס של שיר למשתתפים, המיקס יכול להיות כל שיר מהdataset של פרוייקט זה, ולאחר מכן להשמיע את ההפרדות בהתאם.

תכן הנדסי מפורט:

התכן המפורט יכלול את כל שלבי הפרוייקט: בחירת רכיבי החומרה, גרסאות תוכנה לספריות Python מתאימות, תיאוריה בסיסית, תיאור האלגוריתם ויישומו בקוד.

דרישות בסיסיות ליישום הפרוייקט:

להלן יפורטו הדירישות הבסיסיות עלייהן מושתת הפרוייקט כאשר הן מחולקות ל 2 מרכיבים עיקריים: חומרה ותוכנה.

דרישות אלו נועדו על מנת להריץ את ה source code עבור פרוייקט 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 לפרוייקט 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 מידיי, 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 הקונפיכורציה רלווני 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 musdb18 sigsep שהוא בעצם ה-dataset של הפרוייקט (הסבר על כך בהמשך).

viii. Museval - חבילת פייתרון לביצוע הערכה על שיעורכי מקורות מופרדים.
ix. Google - קישוריות למנוע חיפוש Google.
x. Protobuf - Protocol buffers בשביל Google.

c. CUDA 9 for NVIDIA GPU.

סביבת עבודה למעבר הגרפי מאת NVIDIA אשר מאפשרת לפתח ולהרית תוכניות מחשב על כרטיס ה – GPU, מיועד בעיקר למשימות עיבוד מקבילי מסיבי.

d. PyCharm.

סביבת העבודה עליו קוד ה Python בנוי.

רשת קונבולוציה:

רשת קונבולוציה (CNN – Convolutional Neural Network) הינה אלגוריתם של Deep Learning שפותח בשנת 2012 ע"י אלכס קרישבסקי (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, 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:

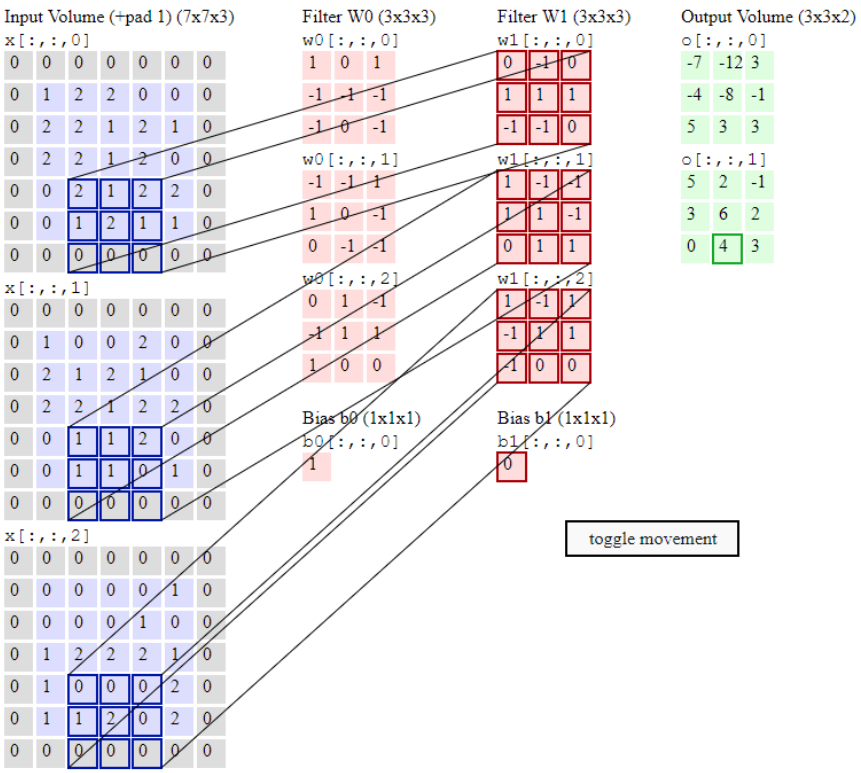
כשמה היא, מבצעים את פעולות הקונבולוציה בין המידע בכניסה לשכבה לבין מספר מסננים כאשר התוצר מכל פעולת קונבולוציה למסנן אחד נקרא Feature map שזה אחד מן המאפיינים של אות המידע.

פעולת הקונבולוציה עובדת באופן הבא:

נתון שאות המידע בכניסה לשכבה מיוצג כמטריצה שגודלה NXM וגודל המסננים הוא KXK, מבצעים כפל איבר באיבר וסכימה (dot product) בין המסנן לבין חתיכה מאות המידע כגודל המסנן KXK 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 $(N-K+1)X(M-K+1)$ שהיא ה feature map שנוצרה עבור המסנן, מכיוון שקיימים מספר מסויים של מסננים נוצרים בהתאם מספר זהה של feature maps, כולן מאפיינים שחולצו מאות המידע ומשמשים כ-output לשכבה הבאה ברשת וככל שברשת יש יותר שכבות קונבולוציה כך נקבלת מאפיינים ברמה יותר גבוהה.

דוגמא לפעולת הקונבולוציה בין אות מידע למסנן והתוצר - feature map:



חשוב לציין שבפרוייקט 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 [5].

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 $Gradient\ Descent$.

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

כאשר:

W_t - וקטור המשקלים באיטרציה t.

γ – קצב הלמידה (Learning Rate).

n – מספר הדגימות.

$Loss\ Function - Q(z, w)$.

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

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

עבור פרוייקט זה נשתמש באופטימייזר ADAM – Adaptive Moment Estimation.

ADAM Optimizer:

אופטימייזר 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: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_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 θ_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)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Pseudo code for ADAM

MUSDB18 Dataset:

זהו ה- Dataset שבו נשתמש בפרוייקט ועל בסיסו מתרחש אימון המודל, הוא מכיל 150 שירים מלאים מסגנונות שונים עם המקורות המופרדים מהם מראש.

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

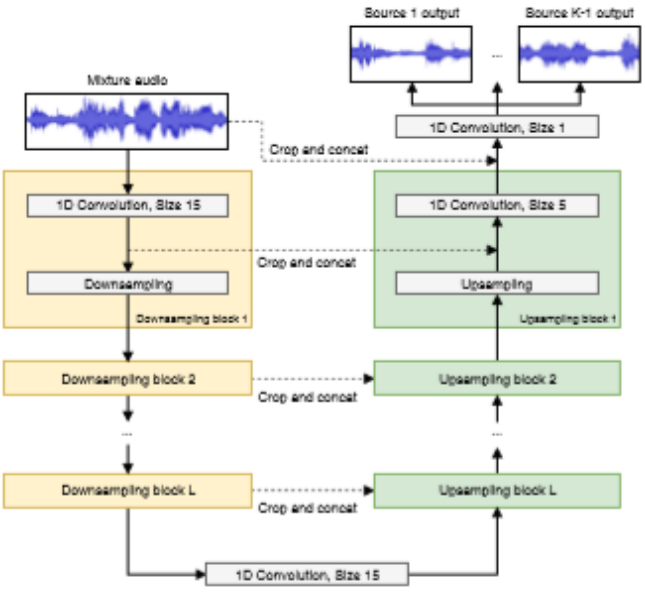
ה- Wave-U-Net הינה מודל לרשת קונבולוציה (CNN – Convolutional Neural Network), המיועד לפתירת משימות של הפרדת מקורות אודיו ופועל ישירות על מקור האודיו במרחב הזמן, מודל זה הוא הבסיס של הפרוייקט מכיוון שהוא עונה על מטרתו המרכזית, להפריד שיר למספר מקורות.

המטרה שלנו היא להפריד מיקס $M \in [-1,1]^{L_m X C}$ ל- K מקורות $S_1 \dots \dots S^k$, כאשר $S^k \in [-1,1]^{L_s X C}$ לכל $k \in \{1 \dots \dots K\}$, 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 שערוכי המקורות K_i [1].

במוצא המודל אנו נקבל את שערוכי 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:



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 רזולוצית הזמן [1].

Up-Sampling Block:

בלוק זה 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].

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 הפרוייקט 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. עלייהם מתבצע חישוב MSE בין המקור המופרד המקורי ב- $Dataset$ לבין השיערוך של המודל כחישוב ה- Loss הכולל של המודל.

עבור הפרוייקט שלנו עפ"י [2] נבחר:

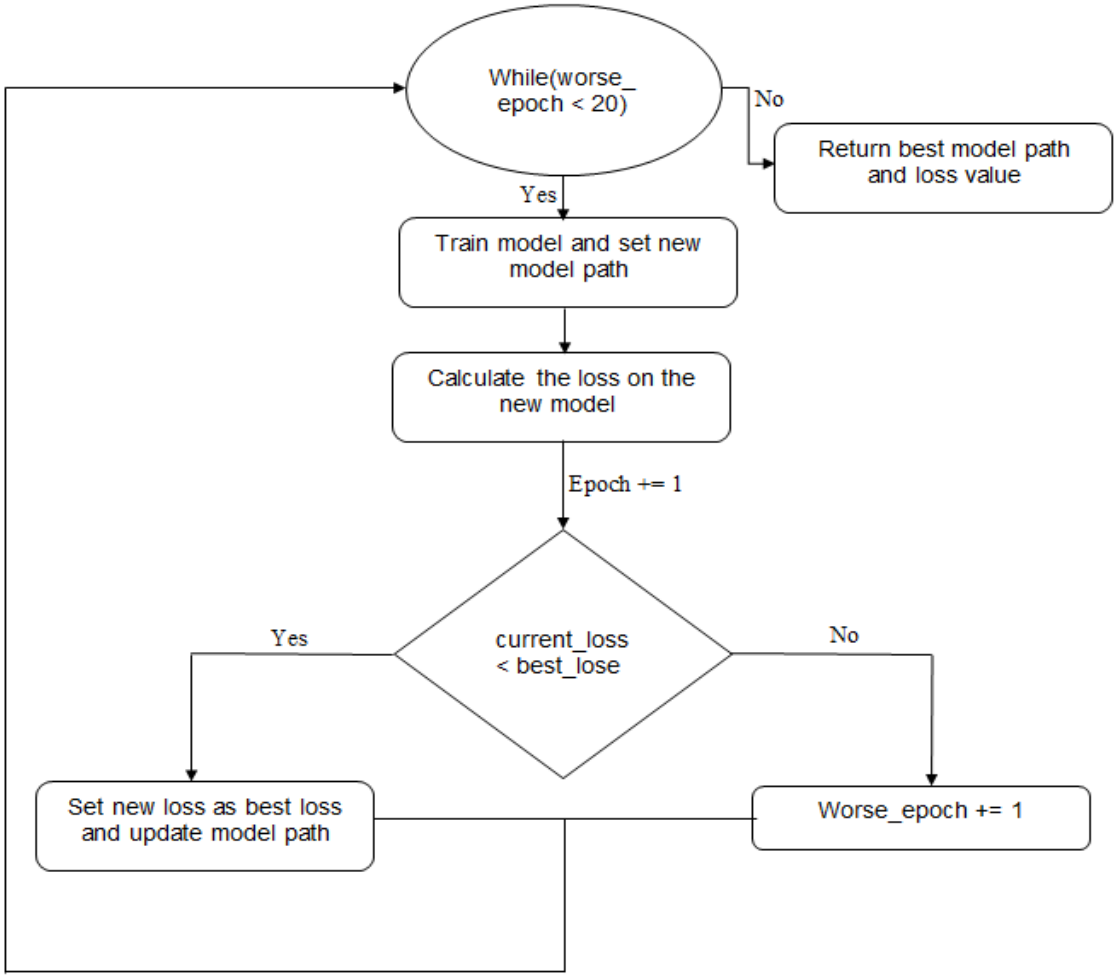
- $L_m = 147443,\ L_s = 16389\ [samples]$.
- $L = 12\ layers$.
- $F_c = 24$ פילטרים נופים עבור כל שכבת קונבולוציה ($F_c^l = l * F_c$).
- $f_d = 15, f_u = 5$ גדלי מטריצת הפילטרים עבור $DS\ Blocks, US\ Blocks$ בהתאמה.
- $Learning\ Rate = 0.0001$.
- $\beta_1 = 0.9, \beta_2 = 0.999$ פרמטרים ל- ADAM.
- $Batch\ Size = 8$ 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:



- **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 וחישוב ההפסד שלו.

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

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

Wave-U-Net architecture block diagram

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

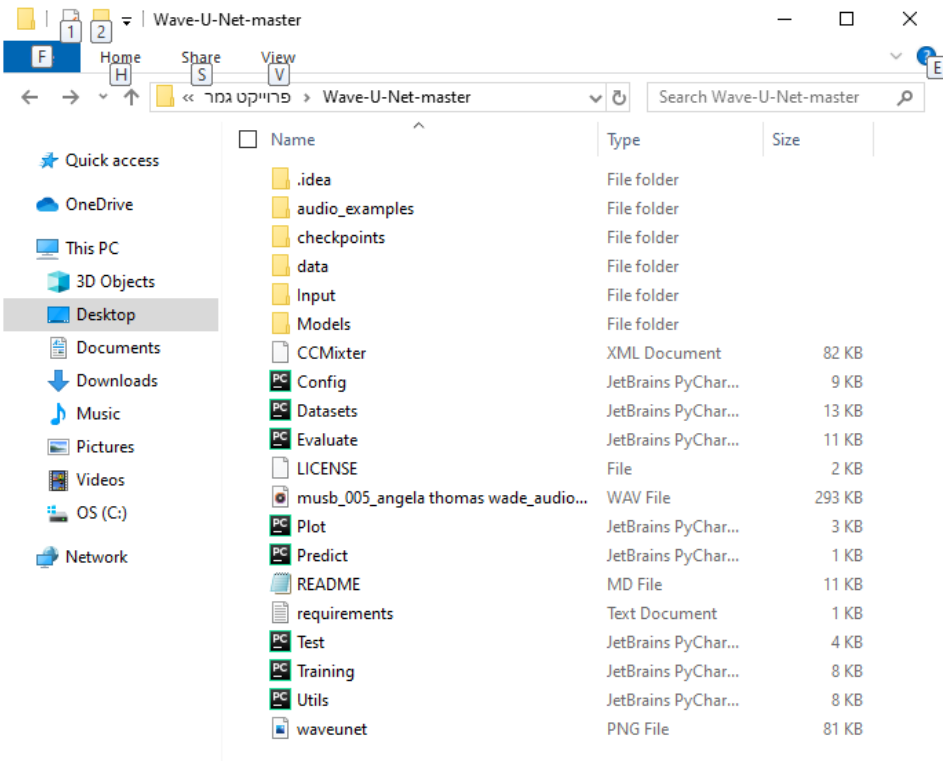
- **Calculate the loss on the new model**: לאחר 2000 איטרציות של בלוק האימון מתבצע חישוב יותר גס של הפסד המודל מאחר ועורכים בדיקה בנוסף על שירים מה- Test set ב- Dataset שהם שירים שלא עברו בתהליך האימון של המודל, שירים אלו ושירים מה- Training set גם כן עוברים במודל בקבוצות כדי לייצר את שיערוכי המקורות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

- **Current_loss < Best_loss**: מתבצעת בדיקה על ההפסד שחושב בבלוק 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 נאתחל את worse_epoch חזרה ל-0 מאחר וקיבלנו שיפור ונבצע עדכון להפסד החדש ושמירת המודל לנתיב המודל הטוב ביותר.

חבילת הקוד של הפרויקט:

פרוייקט 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 הקוד של הפרוייקט:



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 הפרוייקט בו מתבצעת מציאת המודל הטוב ביתר 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 ואחראי על תהליך בניית המודל ומפיק במוצאו את שיערוכי המקורות.
7. Evaluate.py - קוד זה אחראי על חישוב השיערוכים של שיר עבור מודל מסויים ובנוסף לחישוב השערוכים 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 שהוא חישוב שיערוך המקור ע"י החסרת סך המקורות המשוערכים מהמיקס השלם $S^K = M - \sum_{j=1}^{K-1} S^j$.

תוצאות ראשוניות:

לאחר הרצת הקוד וקבלת המודל הטוב ביותר נשתמש בו כדי להפריד שירים מה-Dataset, מכיוון שקיימים ב-Dataset קבצים עם המקורות מופרדים ניתן להשתמש בהם כ-Reference לשיערוכי המקורות שנקבל מהמודל ונבצע בינהם חישוב MSE באופן 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 למול ה-Reference, בגלל שזה חישוב MSE נעלה את ההפרש בריבוע נסכום את כל התואות ונחלק בסך כל הדגימות כדי לקבל את ממוצע של השגיאה.

נבצע פעולה זו על כל אחד מן המקורות (זמר/ת, תופים, בס, צלילים אחרים) באמצעות הקוד הבא:

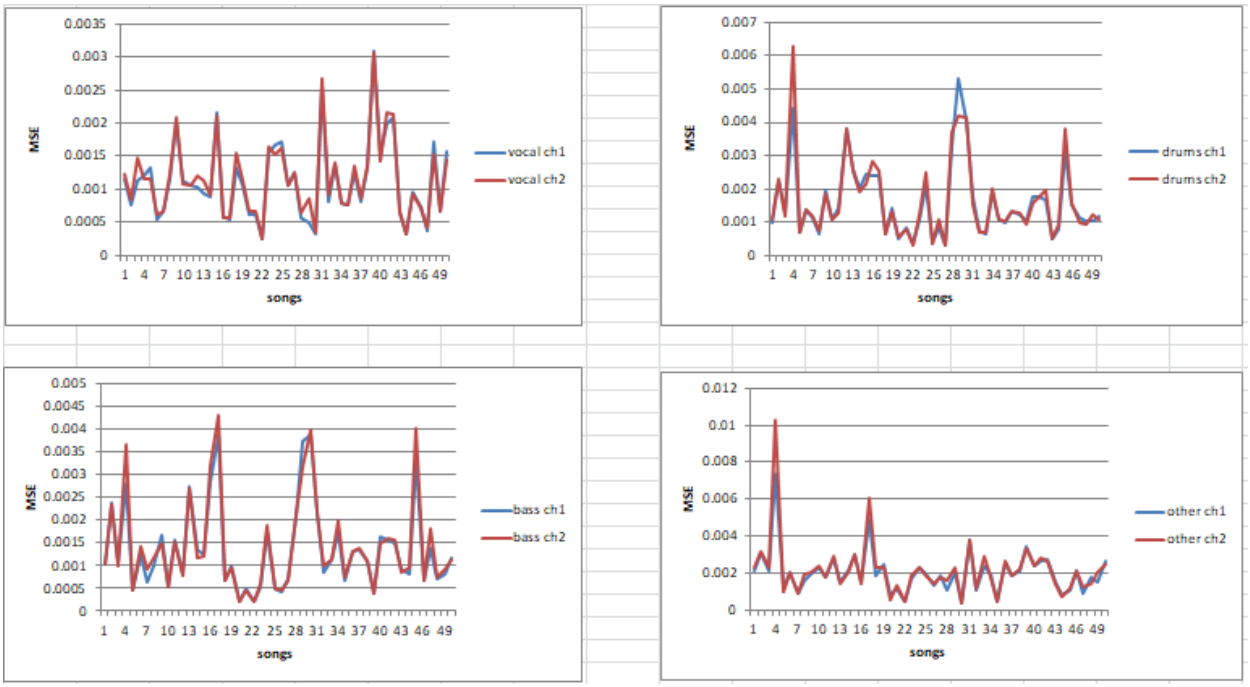
```
import soundfile as sf
import numpy as np
ref, fs_ref = sf.read( 'D:\Ben\FinalProject\Github\Train-test-
dataset\Test\Little Chicagos Finest - My Own.stem_other.wav') # input
- reffrence mixture path
pred, fs_pred =
sf.read('D:\Ben\FinalProject\TestSongs_Test\_other\Little Chicagos
Finest - My Own.stem_mix.wav_other.wav') # input - prediction from U-
Net

mse = np.sum((ref - pred)**2, axis=0) / len(pred) # calculation of
MSE

if len(mse) > 1:
```

```
    print("MSE for channel 0 = {}\nMSE for channel 1 =
{}".format(mse[0],mse[1]))
else:
    print("MSE = {}".format(mse))
```

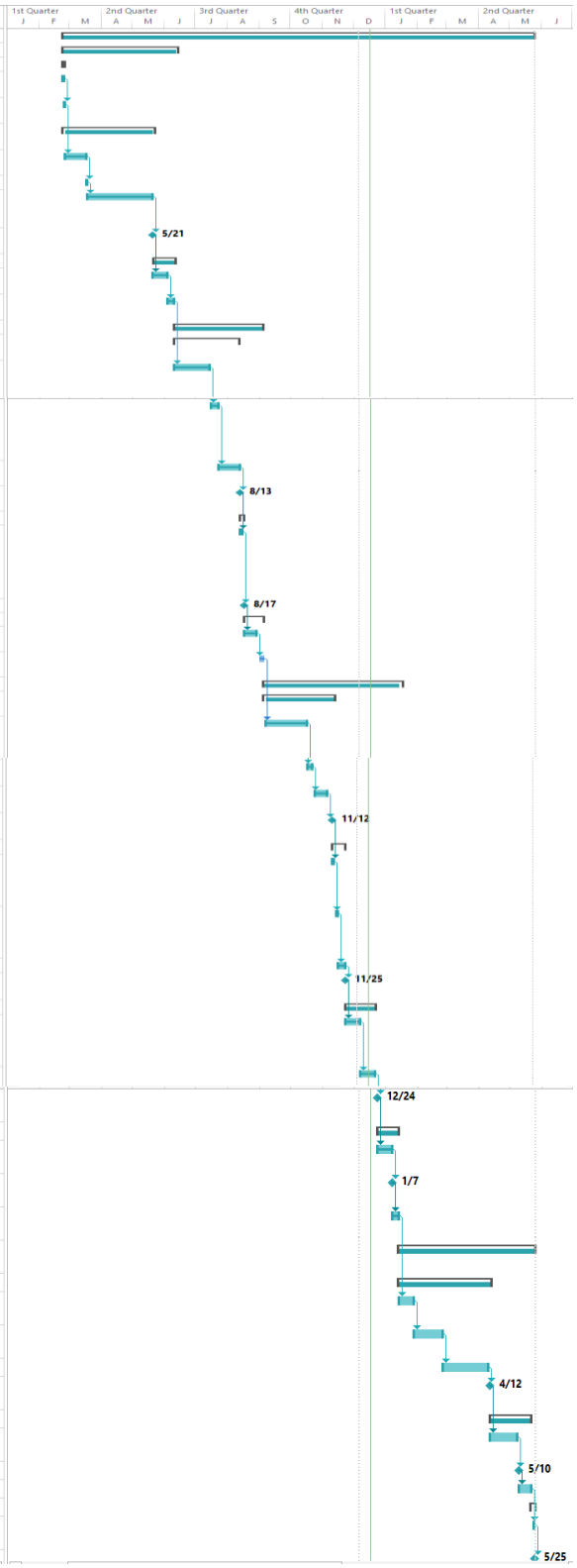
התוצר הוא חישוב MSE ב- 2 ערוצים של mono ו-stereo, ככל שערך זה נמוך יותר כך קיים יותר דמיון בין השערוך למקור.



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

תוכנית עבודה סופית:

		Task Mode	Task Name	Duration	Start	Finish	Predecessors
1			**פרויקט הגמר**	**327 days**	**Mon 2/24/2(**	**Tue 5/25/21**	
2	✓		**SOW**	**81 days**	**Mon 2/24/2(**	**Sun 6/14/20**	
3	✓		**בחירת פרויקט**	**2 days**	**Mon 2/24/2(**	**Tue 2/25/20**	
4	✓		בחירת פרויקט ממאגר	1 day	Mon 2/24/20	Mon 2/24/20	
5	✓		פגישה עם המנחה	1 day	Tue 2/25/20	Tue 2/25/20	4
6	✓		**השלמות פערים וחקר**	**65 days**	**Mon 2/24/20**	**Fri 5/22/20**	
7	✓		בחירת רכיבים למחשב	15 days	Wed 2/26/20	Tue 3/17/20	5
8	✓		קניית מחשב	1 day	Wed 3/18/2(	Wed 3/18/2(	7
9	✓		השלמת קורס הגע ללמידה עמוקה	45 days	Thu 3/19/20	Wed 5/20/20	8
10	✓		פגישה עם המנחה	0 days	Thu 5/21/20	Thu 5/21/20	9
11	✓		**כתיבת הדו"ח**	**15 days**	**Fri 5/22/20**	**Thu 6/11/20**	
12	✓		שליחת טיוטה ראשונית	10 days	Thu 5/21/20	Wed 6/3/20	10
13	✓			5 days	Thu 6/4/20	Wed 6/10/20	12
14	✓		**דו"ח התקדמות**	**62 days**	**Thu 6/11/20**	**Fri 9/4/20**	
15	✓		**השלמות פערי ידע**	**45 days**	**Thu 6/11/20**	**Thu 8/13/20**	
16	✓		השלמת קורס רקע ללמידה עמוקה	24 days	Thu 6/11/20	Tue 7/14/20	13
17	✓		הורדת תוכנות מתאימות Anaconda (Python) ותוספות	5 days	Thu 7/16/20	Wed 7/22/20	16
18	✓		העמקה במאמר	15 days	Thu 7/23/20	Wed 8/12/20	17
19	✓		פגישה עם המנחה	0 days	Thu 8/13/20	Thu 8/13/20	18
20	✓		**GitHub**	**2 days**	**Thu 8/13/20**	**Mon 8/17/2(**	
21	✓		musdb18: הורדה מהאינטרנט של ה-dataset לפרוייקט	2 days	Thu 8/13/20	Fri 8/14/20	19
22	✓		source code	0 days	Mon 8/17/2(	Mon 8/17/2(	21
23	✓		**כתיבת הדו"ח**	**15 days**	**Mon 8/17/2(**	**Fri 9/4/20**	
24	✓		שליחת טיוטה ראשונית	10 days	Mon 8/17/20	Fri 8/28/20	22
25	✓		תיקונים וסיום הדו"ח	5 days	Mon 8/31/20	Fri 9/4/20	24
26	✓		**דו"ח ביניים**	**97 days**	**Fri 9/4/20**	**Sun 1/17/21**	
27	✓		**השלמת פערי ידע נוספים**	**50 days**	**Fri 9/4/20**	**Thu 11/12/20**	
28	✓		CNN - Convolutiona Neural Networks	30 days	Mon 9/7/20	Fri 10/16/20	25
29	✓		ADAM optimizer	5 days	Mon 10/19/20	Fri 10/23/20	28
30	✓		Wave-U-Net Algorithm	10 days	Mon 10/26/20	Fri 11/6/20	29
31	✓		פגישה עם המנחה	0 days	Thu 11/12/20	Thu 11/12/20	30
32	✓		**source code**	**9 days**	**Thu 11/12/2(**	**Wed 11/25/2**	
33	✓		עדכון Python packages לפי קובץ requirement	2 days	Thu 11/12/20	Fri 11/13/20	31
34	✓		התאמה בין חומרה לתוכנה - שינוי פרמטרי קונפיגורציה	2 days	Mon 11/16/20	Tue 11/17/20	33
35	✓		Debuging	5 days	Wed 11/18/2	Tue 11/24/2(	34
36	✓		פגישה עם המנחה	0 days	Wed 11/25/20	Wed 11/25/20	35
37	✓		**ניתוח ראשוני**	**22 days**	**Wed 11/25/2**	**Thu 12/24/2(**	
38	✓		הפרדת שירים מה-dataset באמצעות המודל	10 days	Wed 11/25/20	Tue 12/8/20	36
39	✓		חישוב MSE בין השירים	11 days	Wed 12/9/20	Wed 12/23/20	38
40	✓		פגישה עם המנחה	0 days	Thu 12/24/20	Thu 12/24/20	39
41	✓		**כתיבת הדו"ח**	**15 days**	**Thu 12/24/2(**	**Wed 1/13/2**	
42	✓		שליחת טיוטה ראשונית	10 days	Thu 12/24/20	Wed 1/6/21	40
43	✓		פגיש עם המנחה	0 days	Thu 1/7/21	Thu 1/7/21	42
44	✓		תיקונים וסיום הדו"ח	5 days	Thu 1/7/21	Wed 1/13/21	43
45			**דו"ח מסכם ומסירת הפרויקט**	**95 days**	**Wed 1/13/21**	**Tue 5/25/21**	
46			**ניתוח שני**	**64 days**	**Wed 1/13/2**	**Mon 4/12/21**	
47			הפרדת שירים מסוג mp3	10 days	Thu 1/14/21	Wed 1/27/21	44
48			חישוב SNR על שירים	20 days	Thu 1/28/21	Wed 2/24/21	47
49			ניתוח פרמטר M	32 days	Thu 2/25/21	Fri 4/9/21	48
50			פגישה עם המנחה	0 days	Mon 4/12/21	Mon 4/12/21	49
51			**סיום הדו"ח**	**30 days**	**Mon 4/12/21**	**Fri 5/21/21**	
52			שליחת טיוטה ראשונית	20 days	Mon 4/12/21	Fri 5/7/21	50
53			שה עם המנחה	0 days	Mon 5/10/21	Mon 5/10/21	52
54			ים וסיום הדו"ח	10 days	Mon 5/10/21	Fri 5/21/21	53
55			**מסירת הפרויקט**	**3 days**	**Fri 5/21/21**	**Tue 5/25/21**	
56			עריכת ערכת קוד לפרוייקט	1 day	Mon 5/24/21	Mon 5/24/21	54
57			מסירה	0 days	Tue 5/25/21	Tue 5/25/21	56



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 Macartney, Tillman Weyde, Improved Speech Enhancment with the Wave-U-Net, Submitted at 2018-11-27.

- [4] Pritish Chandna, Audio Source Separation Using Deep Neural Networks, submitted at 2014.

- [5] 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.

- [6] Diederik P.Kingma, Jimmy Lei Ba, ADAM: a method for stochastic optimization, Published as a conference paper ar ICLR 2015.

נספחים:

נספח א' – מפרט מערכת CPU + GPU:

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

נספח ב' – Python Code:

Config.py:

```
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"])
.minimize(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.p
adding) # 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
```