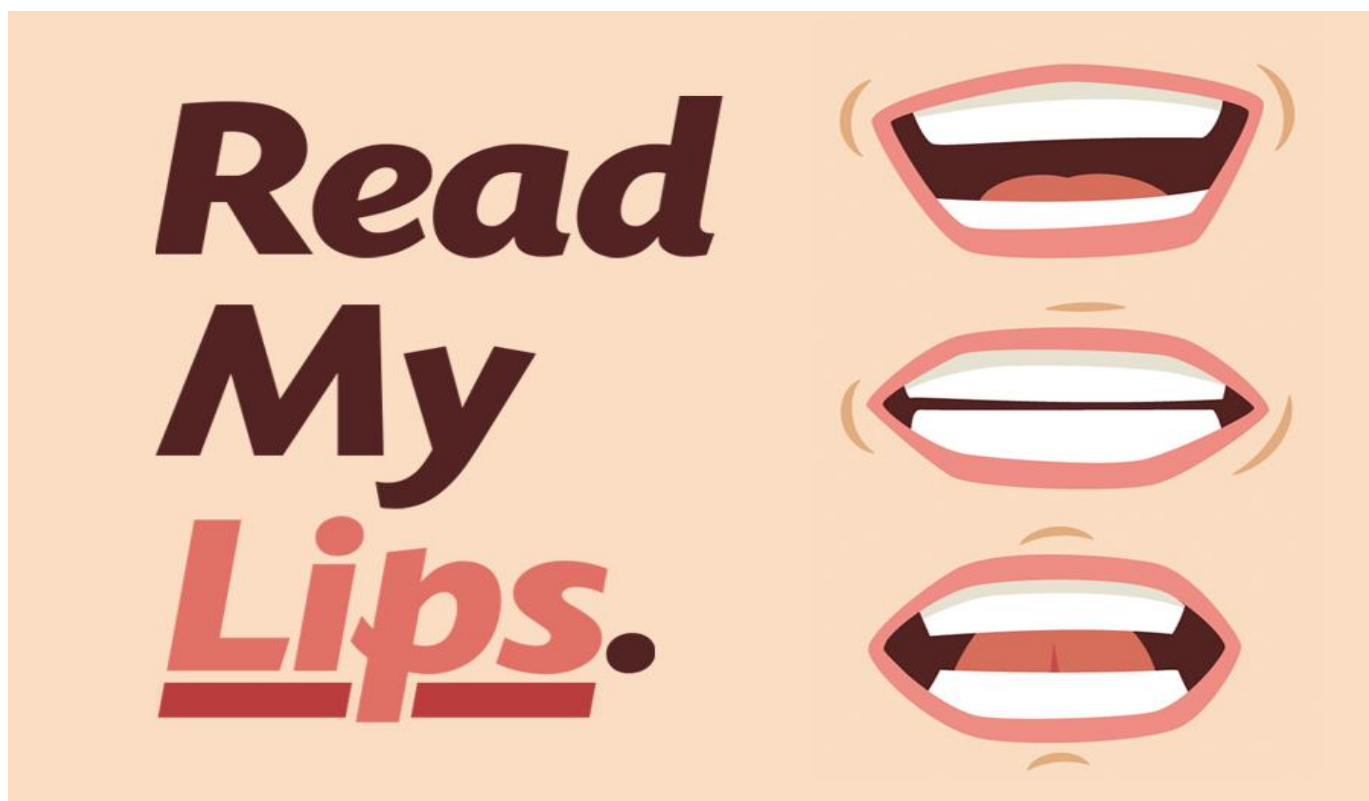


דו"ח פרויקט גמר-

קריאת שפתיים והמרה לטקסט, ומטקסט
לקול

שם הפרויקט: "Read My Lips"



מכללת אפקה להנדסה בתל אביב
מחלקת מדעי המחשב

מגישים:

אורי מטרסו 208959122
שגיא הרשקו 313333619

שם המנחה: ד"ר דינה גורן-בר

תאריך הגשה: 18.4.2023

תודות

תודה לד"ר דינה גורן-בר על הליווי המסור וההנחיה לאורך כל עבודת הפרויקט שלנו. על העזרה הרבה בהכוונה למחקרים ומקורות מידע רבים אשר תרמו לנו מאוד והאמונה בנו גם כשחשבנו שהפרויקט מורכב מדי לביצוע.

תודה לעומר נחשון על הליווי במהלך הפרויקט, הזמינות הרבה ותמיד שם כדי לתת מענה.

לכותבי המאמר:

LIPNET: END-TO-END SENTENCE-LEVEL LIPREADING

Yannis M. Assael , Brendan Shillingford , Shimon Whiteson & Nando de Freitas Department of Computer Science, University of Oxford, Oxford, UK Google DeepMind, London, UK 2 CIFAR, Canada 3

לכותבי המאמר:

Deep Voice: Real-time Neural Text-to-Speech

Sercan O' . Arik, Mike Chrzanowski, Adam Coates ,Gregory Diamos ,Andrew Gibiansky, Yongguo Kang , Xian Li , John Miller , Andrew Ng Jonathan Raiman , Shubho Sengupta , Mohammad Shoeybi

תוכן עניינים

2	תודות
4	תקציר
7	מבוא – מוטיבציה והגדרת הבעיה
8	מבוא - מטרות ויעדים
9	סקירת ספרות
12	סקר שוק - ניתוח מתחרים
15	חלופות
17	ארכיטקטורה
20	תכן מפורט
23	תיאור התוצר גרסת ALFHA - אלגוריתמים
27	קוד
39	הדגמה
42	הערכה- DATA SET
43	מדדים
44	צורת הבדיקה
46	תוצאות
47	סיכום ומסקנות
48	רשימת מקורות

תקציר

הפרויקט שלנו עוסק בפענוח תנועות שפתיים והמרתו לטקסט ומשם לקול כך שנוכל להנגיש ולהקל על אוכלוסיית החירשים, כבדי השמיעה והאילמים בהבנת תקשורת בין אנשים ולגשר על פערי הלקות עם אוכלוסיית השומעים.

ביצענו סקירת ספרות מעמיקה סביב נושא קריאת השפתיים. ראינו כי היכולת של קריאת שפתיים מקשה על חייהם של בעלי הלקויות משום שהדבר מצריך מהם מאמץ רב בתרגום השפתיים שלא תמיד יוצא מדויק. במאמרים ראינו כי קיים פתרון לבעיה זו והחלטנו להביא למימוש מערכת זו. כמו כן, במהלך המחקר עלה בדעתנו לשלב בפרויקט את המרת הטקסט לקול ובכך להנגיש את המערכת לאוכלוסיית האילמים ולהשמיע את קולם וכן לאוכלוסיית השומעים שלהם ישנה האפשרות לקרוא ואף לשמוע את הנאמר מאנשים בעלי הלקות.

כמו כן, ביצענו ניתוח מתחרים וראינו מגוון פתרונות מעניינים בתחום של קריאת השפתיים ובתחום טקסט לקול. אולם, ראינו כי אף אחד מן המתחרים אינו מציע את הפתרון שלנו שנחוץ עבור קהילה רחבה יותר של אנשים בעלי לקויות זאת מאחר ואנו משלבים בין שני האלגוריתמים.

בשלב הבא ביצענו תכנון מפורט של המערכת שלנו. החלטנו לכלול שני אלגוריתמים למערכת: האלגוריתם הראשון, מבצע פענוח של תנועות השפתיים ע"י זיהוי אזור הפנים וחילוץ אזור השפתיים. האלגוריתם מפצל את הסרטון לפריימים, בכל פריים הוא חוזה את ההברה ויוצר רצף שלהם. בעזרת רצף זה הוא מעבד אותם לאותיות ואוסף אותם למילים ויוצר משפט. האלגוריתם השני, ממיר את הטקסט המעובד ובעזרת סנתוז מפרק את האותיות להברות ופעולת נוספות שמתבצעות ומשם לקול (פירוט מלא בסקירת הספרות).

בשלב הפיתוח של המערכת קיבלנו תוצר אשר מומש בשפת התכנות Python ובשילוב GRID dataset המכיל סרטונים של מספר דוברים וקבלת טקסט של הנאמר בסרטונים הללו. בשילוב עם אלגוריתם TTS אשר נכתב בשפת Python ועושה שימוש בקריאות API לשרתי IBM ומבצע את המרת הטקסט לקול.

מדדנו את אחוז הדיוק של המערכת ע"י: CER (character error rate) שזה אחוז השגיאות בחיזוי האותיות במילה, ו-WER (word error rate) שזה אחוז השגיאות בחיזוי המילים במשפט. וקיבלנו WER בתוצאה של 12% ו-CER בתוצאה של 7%.

תוצאות אלו מראות שאכן המערכת שלנו מצליחה בצורה מרשימה לחזות את הנאמר ע"י הדובר ולייצר פלט טקסט וקול נכונים עם אחוז שגיאות נמוך.

ABSTRACT

Our project deals with decoding lip movements and converting it to text and from there to voice so that we can make it accessible and easier for the deaf, hard of hearing and mute population to understand communication between people and to bridge the disability gap with the hearing population.

We performed an in-depth literature review on the topic of lip reading. We have seen that the ability to read lips makes the lives of those with disabilities difficult because this requires a great deal of effort from them in lip translation which is not always accurate. In the articles we saw that there is a solution to this problem and we decided to implement this system. Also, during the research, it occurred to us to incorporate into the project the conversion of text to voice, thus making the system accessible to the mute population and making their voices heard, as well as the hearing population having the opportunity to read and even hear what is being said by people with disabilities.

We also performed a competitor analysis and saw a variety of interesting solutions in the field of lip reading and text to voice. However, we have seen that none of the competitors offers our solution which is necessary for a wider community of people with these disabilities since we combine the two algorithms.

In the next step we carried out a detailed planning of our system. We decided to include two algorithms in the system: the first algorithm, decodes the movements of the lips by identifying the face area and extracting the lip area. The algorithm splits the video into frames, in each frame it predicts the syllable and creates a sequence of them. Using this sequence it processes them into letters and collects them into words and creates Sentence.

The second algorithm, converts the processed text and, with the help of synthesis, breaks down the letters into syllables and additional operations are carried out and from there into sound (full details in the literature review).

During the development phase of the system, we received a product that was implemented in the Python programming language

and combined GRID dataset containing videos of several speakers and receiving text of what is said in these videos. In combination with a TTS algorithm which is written in the Python language and uses API calls to IBM servers and performs the text to voice conversion.

We measured the accuracy percentage of the system by: CER (character error rate) which is the percentage of errors in predicting the letters in a word, and WER (word error rate) which is the percentage of errors in predicting the words in a sentence. And we got a WER with a result of 12% and a CER with a result of 7%.

These results show that indeed our system is impressively successful in predicting what is said by the speaker and producing correct text and voice output with a low percentage of errors.

מבוא – מוטיבציה והגדרת הבעיה

- בישראל ישנם כ-8000 חירשים וכ-500 אלף כבדי שמיעה אשר נעזרים בשפת הסימנים ובקריאת שפתיים כאמצעי תקשורת. בארה"ב כ-3 אחוזים מהאוכלוסייה הם חירשים (כ-10 מיליון איש).
- בנוסף, 1.5% מאוכלוסיית העולם אשר אינם יכולים להביע את עצמם באמצעות דיבור.
- גם אנשים בעלי שמיעה תקנית נעזרים בתנועות שפתיים על מנת להבין את הנאמר. אולם, ישנם קוראי שפה אשר נולדו כחרשים ומתקשים יותר בפענוח המידע הוויזואלי בקריאת דיבור ולא מצליחים לקלוט את כל המידע מהצד השני.

מבוא - מטרות ויעדים

מטרתה של המערכת שלנו היא פענוח תנועות שפתיים והמרתו לטקסט ומשם לקול כך שנוכל להנגיש ולהקל על אוכלוסיית החירשים, כבדי השמיעה והאילמים בהבנת תקשורת בין אנשים ולגשר על פערי הלקות עם אוכלוסיית השומעים.

מטרות ויעדים:

-הנגשת המערכת ללקוי השמיעה והדיבור.

-יצירת תקשורת שווה בין אוכלוסיית השומעים לאוכלוסיית לקויי השמיעה והדיבור.

- נתינת קול ללקויי דיבור ויצירת הבנה ללא שפת סימנים וקריאת שפתיים מאומצת ללקויי השמיעה.

-שיפור יכולות המערכת ע"י שילוב שני אלגוריתמים ליצירת פלטי טקסט וקול מתנועות שפתיים.

סקירת ספרות

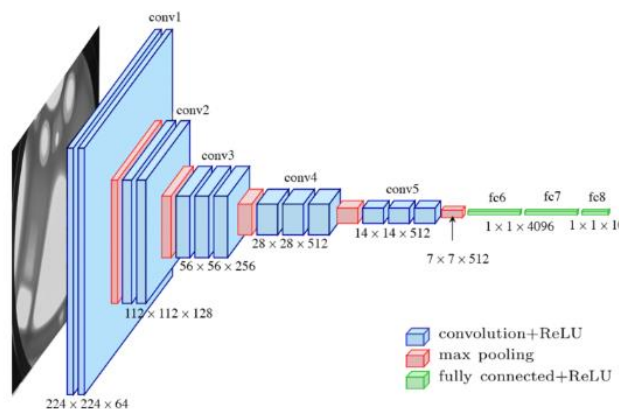
קריאת שפתיים היא טכניקה של הבנה ופירוש חזותי של התנועות בשפתיים ללא קול. ביכולת זו עושים שימוש אנשים עם לקויות שמיעה כך שהם יוכלו להבין את הנאמר בשיחה עם אדם אחר.

באמצעות קריאת שפתיים ניתן להבין יותר בקלות תהליכים כמו:
-קליטת שפה דרך תווי פנים, שפתיים ועיניים.
-הבנת מידע המדובר בתקשורת לא ורבולית.
-יצירת "סנכרון" בין הנאמר לסביבה.

בפרויקט זה אנו נעשה שימוש בשני אלגוריתמים ונדאג לשילובם בכדי לממש את מטרותנו.
לפניכם שלושה אלגוריתמים הפותרים בעיה זו:

1. אלגוריתם פענוח תנועות שפתיים והמרה לטקסט:

אלגוריתם ראשון- VGGNet (Visual Geometry Group networks) **היא** רשת קונבולוציה של רשת נוירונים המשתמשת בפעולת קונבולוציה. קונבולוציה היא פעולה בינארית בין שתי פונקציות או סדרות ערכים, רשת זו מאומנת לקלוט מאות אלפי תמונות ולסווג אותם באמצעות פילטרים, היא מורכבת מ-16 שכבות וכל שכבה ברשת מחלצת פיצ'רים רבים כמו גבולות התמונה ותבניות הפנים וכך מזוהה תנועת השפתיים. לאחר חילוץ התבניות מכונת הלמידה חוזה את ההברות הנאמרות, ובעזרת ארכיטקטורת רשת נוירונים LSTM (Long short-term memory) מחברת אותם למילים ומשפטים ולבסוף פלט טקסט של הנאמר בסרטונים. ה- data set שעליו נבדק האלגוריתם הכיל סרטונים של אנשים מבטאים מספרים מאחד עד תשע והוא נמצא מדויק ב-88 אחוזים וטוב יותר ב-3 אחוזים בהשוואה למודל CNN-RNN רגיל.



חסרון של האלגוריתם זמן העיבוד שלו אורך זמן רב בהשוואה לאלגוריתם LipNet אותו נציג בהמשך. [1]

אלגוריתם שני- LipNet היא ארכיטקטורת רשת עצבית לקריאת שפתיים הממפה רצפים באורך משתנה של וידאו, מסגרות לרצפי טקסט, והוא מאומן מקצה לקצה.

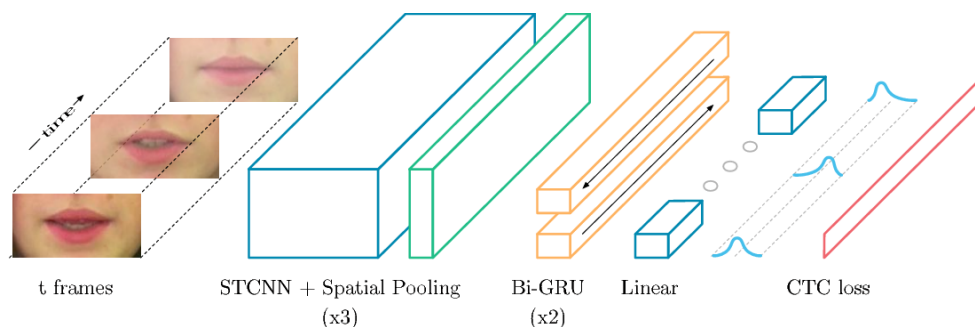
הוא מורכב מ-3 ערוצים המייצגים זמן, תמונה, וקבוצת פיקסלים של התמונה המצולמת.

רשת הנוירונים קולטת את תנועות השפתיים לאורך זמן דרך קונבולוציה תלת מימדית מסוג STCNN ע"י חילוף תבנית השפתיים והגבולות שלה.

משם נשלחים התבניות אל Bi-GRUs שהוא ארכיטקטורת רשתות נוירונים ותפקידו באלגוריתם לקרוא שפתיים משני כיוונים כלומר מתחילתו ומסופו ובכך לאמן את מכונת הלמידה במהירות ולקבלת תוצאות מהירות יותר.

לבסוף, טרנספורמציה ליניארית מוחלת בכל שלב זמן, ולאחר מכן מתבצעים חישובי סטטיסטיקה שבסופם ע"י חישובי CTC מתבצע חיזוי והתאמה בין תנועות השפתיים לפירושן כפלט טקסט.

בבדיקה שנעשתה על GRID dataset נמצא כי האלגוריתם מגיע לרמת דיוק של 95 אחוז בזיהוי קריאת שפתיים. [2]



אלגוריתם שלישי- data set של סרטונים באיכות נמוכה המכילים תנועות חדות.

אלגוריתם זה מפרק את סרטון הקלט לפריימים ומשפר אותם על ידי יישום שיטת ניגודיות מוגבלת אדפטיבית וחישובים סטטיסטיים. לאחר מכן, פרצופים מזוהים ע"י פרונטליזציה של הפנים

Generative Adversarial Network (FF-GAN). לאחר מכן, אזור הפה מחולץ.

אזור הפה שחולץ בסרטון כולו והנאמר ממנו מועברים לפיענוח ResNet במהלך תהליך הלמידה.

אלגוריתם זה נבדק על ה LRW data set המכיל סרטוני פנים כך שהפנים שאינן מיושרות פוענחו מול פנים לאחר יישור על מנת להגדיל את גודל מסד הנתונים.

שיטה זו מגיעה לחיזוי מדוייק בהשוואה לאלגוריתמים אחרים של פיענוח קריאת שפתיים והוא נמצא מדויק ב-84 אחוזים. [3]

2. אלגוריתם המרה מטקסט לקול (TTS- text to speech):

טקסט לדיבור (tts) הוא תוצר של מערכת הבנויה מרשתות נוירונים עמוקים, המורכבת מחמישה בלוקים. WaveNet הוא גנרטור משופר בהשוואה לגנרטורים אחרים בתחום ההמרה מטקסט לקול, וכן מבצע זאת עם פרמטרים מועטים ובמהירות גבוהה יותר. [4].

השלבים מקבלת קלט הטקסט עד לקבלת פלט הקול הם:

1. grapheme-to-phoneme conversion model:

מודל זה ממיר את הטקסט לאותיות קידוד ע"פ פורמט מקודד של אותיות.

2. segmentation model for locating phoneme boundaries:

המודל מזהה את גבולות ההברה באוסף קבצי השמע.

ממיר את ההברה ומאבחן היכן היא מתחילה והיכן מסתיימת.

3. phoneme duration prediction model:

מודל החוזה את אורך ההברה ברצף האותיות שבסופו מתקבלת מילה.

4. fundamental frequency model:

מודל החוזה את אורך תדר ההברה במילה.

5. audio synthesis model:

מודל זה משלב בין ארבעת המודלים ומאחד את כל ההברות לפלט קול של מילה שלמה.

האלגוריתם נבדק ע"י אימון המודלים על מאגר דיבור פנימי באנגלית המכיל כ-20 שעות של נתוני דיבור מפולחים לתוך 13,079 התבטאויות. ולעומת אלגוריתמים אחרים הגיע לכפול 400 במהירות. [4]

הפיתרון שלנו:

אנו נשתמש באלגוריתם LipNet ובאלגוריתם TTS ונשלב ביניהם על מנת ליצור אינטגרציה בין השפתיים-לטקסט ומטקסט-לקול כך שהאלגוריתם שלנו ייצר משפתיים-לקול.

בחרנו ב-LipNet מפני שאחוזי ההצלחה שלו הגבוהים ביותר, ובחרנו ב-TTS מפני שהוא הכי מהיר מבין שאר האלגוריתמים.

סקר שוק – ניתוח מתחרים



LipSight - חברת סטארטאפ המספקת שירותי קריאת שפתיים העושה שימוש בבינה מלאכותית שמסוגלת לאבחן שפה ע"י זיהוי תנועות פנים מוידאו והמרתו לטקסט ומשמשת לפקודות קוליות ברכב ולאבטחה.



Lipifai - היא טכנולוגיה לקריאת שפתיים המסייעת לאנשים חירשים וכבדי שמיעה לתקשר עם סביבתם, במיוחד בסביבות רועשות. על ידי שימוש בבינה מלאכותית, Lipifai ממירה תנועות שפתיים וקול לתמלול טקסט בזמן אמת.



Liopa - חברה העוסקת בפיתוח זיהוי ויזואלי של קול (vsr). זיהוי הנאמר דרך תנועות השפתיים מתבצע ע"י אבחון תנועת השפתיים של הדובר למצלמה. כמו כן, החברה מפתחת מערכת קריאת שפתיים אשר מסוגלת לפענח את הנאמר גם בסביבה רועשת.



Speechify - חברה אשר פיתחה תוכנה העובדת בכמה פלטפורמות והיא ממירה טקסט לקול. התוכנה עוזרת לקרוא טקסטים מהר יותר ברשת ובאפליקציות, תומכת במגוון שפות ואף עוזרת להנגיש טקסט לאנשים בעלי דיסלקציה.



Speechelo -המרה מיידית מטקסט דרך שירותי ענן עם אפשרויות רבות לקולות שונים (מעל 30 קולות אנושיים לשני המינים). התוכנה מסוגלת להקריא בטונים שונים, תומך ב-24 שפות ותמיכה בתוכנות עריכה ווידאו.



Notevibes - תוכנה להמרת קול לטקסט באמצעות בינה מלאכותית אשר משתמשת בקולות אנושיים מציאותיים במגוון שפות. ניתן להשתמש בתוכנה זו על מנת ליצור סרטוני לימוד, מכירות והדרכה. עושה שימוש באבטחה מודרנית למניעת דליפת מידע.

התובנות שלנו:

כיום לא קיימת מערכת שדומה למערכת שלנו . קיימות בשוק כמה חברות העוסקות בקריאת שפתיים והמרתו לטקסט וקיימות חברות העוסקות בהמרת טקסט לקול, אך השילוב של שניהם עדיין לא קיים. אנו נבנה אלגוריתם משולב המחבר בין שני האלגוריתמים כך שגם כבדי ראייה, שמיעה ודיבור יוכלו להבין את הנאמר מן הדובר באמצעות הפתרונות הללו.

אנו מאמינים כי לקות שמיעה ודיבור הן לקויות אשר ניתן להתגבר עליהן, ובעזרת האמצעים הנכונים נוכל לגשר על פערים אלו עם אוכלוסיית השומעים. קריאת שפתיים היא מיומנות שלא כולם יודעים אותה, היא מיומנות יותר בקרב אוכלוסיית לקויי שמיעה על מנת לפענח שפה ע"י אספקטים ויזואליים ומצריכה מאמץ רב.

אנו מקווים שהאלגוריתם המשופר יעזור לאוכלוסייה זו לתקשר בקלות ובמהירות ללא מאמץ, ויספק קול לאילמים ואנשים בעלי קשיי דיבור.

שם חברה קריטריונים/	LipSight	Lipifai	Liopa	Speechify	Speechelo	Notevibes	Our project
קריאת שפתיים	כן	כן	כן	לא	לא	לא	כן
טקסט לקול	לא	לא	לא	כן	כן	כן	כן
אוכלוסיית יעד	כלל האוכלוסייה	לקויי שמיעה וחירשים	לקויי דיבור/שמיעה וחירשים	כבדי ראייה, דיסלקטים	כלל האוכלוסייה	כלל האוכלוסייה	לקויי דיבור/ראייה/ שמיעה וחירשים
אמצעים	מצלמה	מצלמות מרובות	מצלמה	רמקול, מסך	רמקול, מסך	רמקול, מסך	מצלמה ורמקול
טכנולוגיות	Artificial intelligent, Deep Learning	Artificial intelligent	Artificial intelligence, VSR	Machine Learning, TTS	TTS	Ai TTS, Cloud services	Artificial intelligent, Machine Learning, TTS , IBM services
אפליקציה	אין	אין	יש	יש	יש	אין	אין

חלופות

VGGNet

ארכיטקטורת רשת נוירונים קונבולוציונית (CNN) פופולרית שיושמה בהצלחה למגוון רחב של משימות ראייה ממוחשבת, כולל זיהוי אובייקטים, סיווג תמונות וזיהוי פנים. עם זאת, כשמדובר בקריאת שפתיים, ישנם כמה יתרונות וחסרונות לשימוש ב-VGGNet:

יתרונות:

- בעלת ארכיטקטורה עמוקה שמאפשרת למידה של עצמים מורכבים מקלט מידע כמו מקריאת תנועות שפתיים וצורות הדיבור שלהם.

- עושה שימוש במספר רב של שכבות קונבולוציה המאפשרות ביצוע חילוץ אלמנטים רבים.

- ארכיטקטורה מובנית בעלת מקורות ומודלים מאומנים רבים אשר זמינים למחקר אודות קריאת שפתיים ואימון.

חסרונות:

- במקור הארכיטקטורה מותאמת לתמונות ולא מותאמת לקריאת שפתיים שהוא תהליך ייחודי ומורכב יותר.

- VGGNet היא רשת גדולה מאוד ויקרה מבחינה חישובית, מה שעלול להקשות על אימון מערכי נתונים גדולים או עם משאבי חישוב מוגבלים.

- קושי בלכידת האופי הדינמי והרציף של תנועות השפתיים בעת דיבור אשר מקשה על תמלול השפה.

:Diverse Pose Lip-Reading(DPLR)

DPLR היא מערכת המשתמשת בטכניקות ראייה ממוחשבת ולמידת מכונה כדי לזהות אוטומטית מילים מדוברות על ידי ניתוח תנועות השפתיים והפנים של הדובר. המערכת משתמשת בקבוצה מגוונת של זוויות והבעות פנים על מנת לשפר את הדיוק שלה בזיהוי מילים, מה שהופך אותה לחזקה יותר בפני מגוון וריאציות בסגנון דיבור ורעשי רקע.

יתרונות:

דיוק משופר: על ידי שימוש במערך מגוון של תנוחות והבעות פנים, DPLR יכול לשפר את הדיוק שלו בזיהוי מילים מדוברות, אפילו בסביבות רועשות או כאשר לרמקול יש סגנון דיבור שונה.

ביצועים בזמן אמת: DPLR תוכנן לעבוד בזמן אמת, מה שהופך אותו לשימושי עבור יישומים כגון זיהוי דיבור בסביבות רועשות או עבור לקויי שמיעה.

גמישות: ניתן לאמן DPLR על שפות ודיאלקטים שונים, מה שהופך אותו לשימושי עבור יישומים בהקשרים תרבותיים שונים.

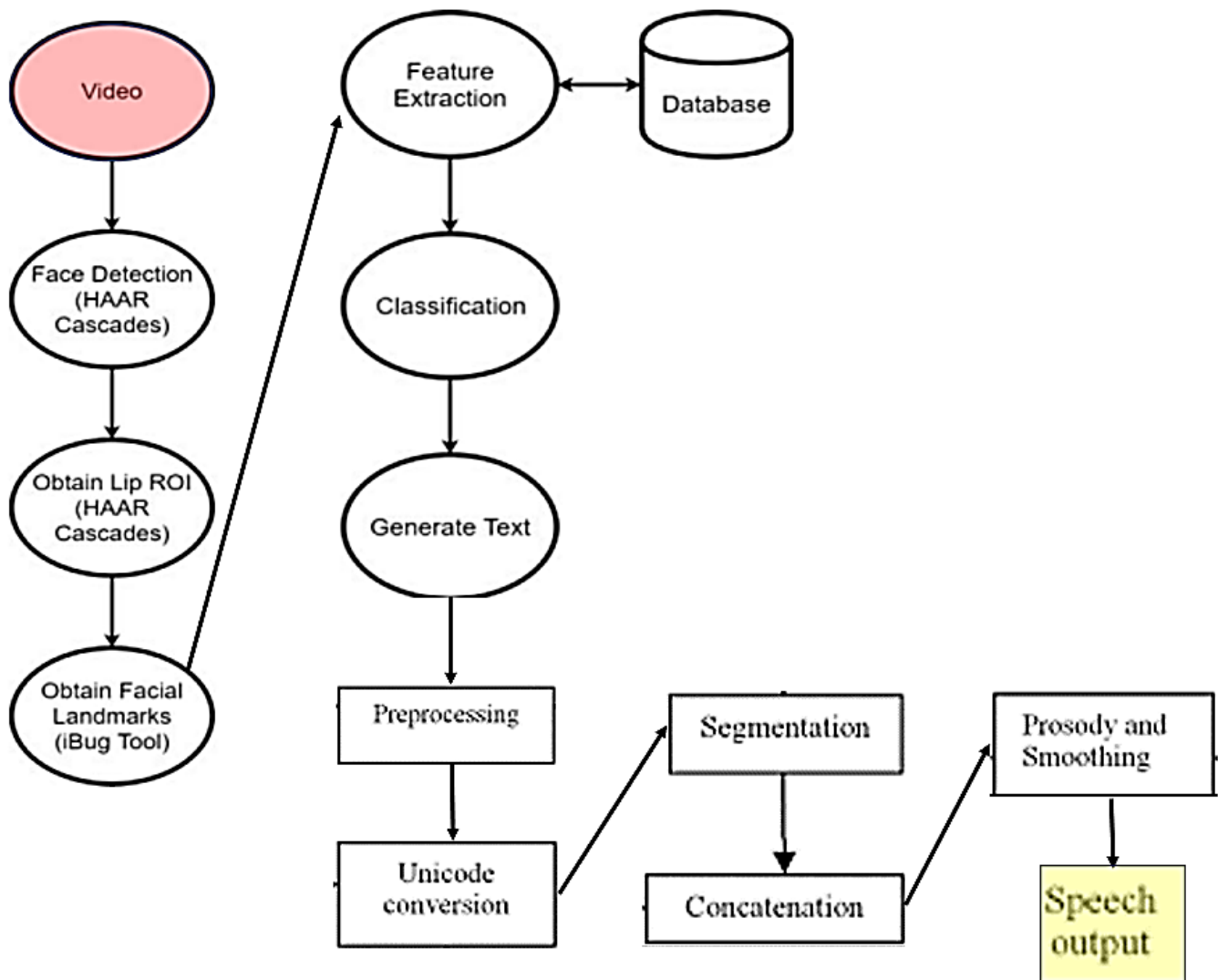
חסרונות:

תלות בנתוני אימון: הדיוק של DPLR תלוי במידה רבה באיכות ובכמות נתוני האימון הזמינים. חוסר גיוון בנתוני האימון עלול להוביל לתוצאות מוטות.

אוצר מילים מוגבל: הדיוק של DPLR בדרך כלל נמוך יותר לזיהוי מילים מחוץ לאוצר המילים האימון שלו, מה שהופך אותו פחות שימושי עבור יישומים הדורשים זיהוי מגוון רחב של מילים או ביטויים.

ארכיטקטורה

תרשים ארכיטקטורה - Block Diagram:



תיאור הארכיטקטורה:

Face detection (HAAR cascades):

אלגוריתם שניתן להשתמש בו למשימות לזיהוי אובייקטים, או משתמשים בו לזיהוי פנים.

האלגוריתם מאמן את המסווג ע"י תמונות רגילות ומנוגדות על מנת לזהות אובייקטים בתמונות חדשות, במקרה שלנו המסווג יודע לזהות תמונות עם פנים וללא פנים בכך שהוא מזהה דפוסי פיקסלים שמרכיבות תבנית פנים.

Obtain lip ROI (HAAR cascades):

זהו תהליך המזהה את מיקומן של השפתיים במסגרת תמונה או וידאו.

Obtain facial landmarks (ibug tools):

זהו תהליך השגת נקודות ציון בפנים כדי לזהות ולסמן נקודות ספציפיות על הפנים המתאימות למאפיינים מרכזיים כגון זוויות הפה, קצה האף וקצוות קו הלסת. לאחר מכן ניתן להשתמש בנקודות אלו ליצירת מפה של הפנים ולבצע עליהן סוגים שונים של ניתוח.

Feature extraction:

הוא תהליך של חילוץ תכונות שימושיות מאוסף תמונות שקיימות לנו בדאטה סט. לאחר מכן ניתן להשתמש בתכונות אלה כדי לאמן מודלים לביצוע משימות כגון סיווג או רגרסיה, אנו נשתמש בסיווג.

Classification:

סיווג הוא טכניקה שבה מודל מאמן להקצות תוויות או מחלקה לנתוני קלט נתון. בארכיטקטורה שלנו הסיווג משמש לניבוי צילי הדיבור על סמך המידע החזותי בסרטון.

תהליך הסיווג כולל אימון מודל על מערך נתונים של דוגמאות מסומנות, כאשר כל דוגמה מורכבת מסרטון של שפתיו של הדובר ותעתיק של המילים המדוברות.

Generate text:

ייצור הטקסט שמפוענח ע"י ה-LSTM (סוג של רשת נוירונים) שקולטת רצפי קלט(הברות) מתוך פריימים של סרטונים, והצגת הטקסט.

Preprocessing:

תהליך זה הוא שלב מקדים והכרחי אשר מכין את קלט הטקסט לשלב הסינתזה באלגוריתם ה-TTS ע"י פירוק המילים ליחידות קטנות והתעלמות מרווחים מיותרים.

Unicode conversion:

קידוד הטקסט על מנת לייצג תווים בטקסט שמומר לדיבור. הקידוד נעשה כדי לקבוע את ההגייה הנכונה ובאמצעותו ניתן להמיר כל טקסט לדיבור.

Segmentation:

המודל מזהה את גבולות ההברה באוסף קבצי השמע. ממיר את ההברה ומאבחן היכן היא מתחילה והיכן מסתיימת.

Concatenation:

מודל זה מאחד את יחידות ההברה למילה ע"י בחירת יחידות שבהם המערכת חוזה את ההתאמה הטובה ביותר להרכבת הפלט הסופי.

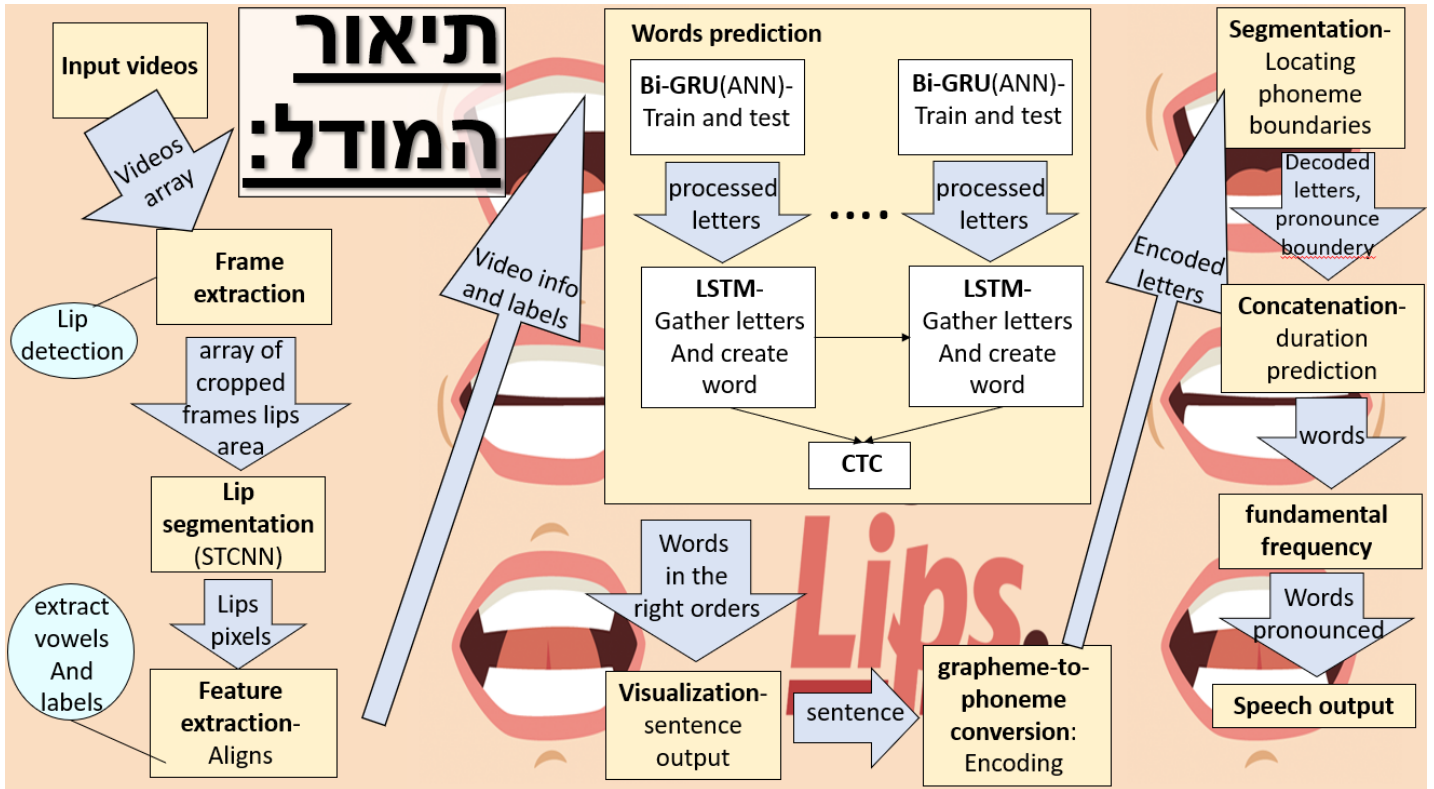
Prosody and smoothing:

במודל זה המילה מסונתזת ע"י שינוי גובה הצליל, משך הזמן והעוצמה של הדיבור כך שיתאימו לרגש המובע ממשמעות הטקסט והפחתת הצליל הרבובטי ויגרום לו להישמע יותר כמו דיבור אנושי.

Speech output:

התוצר הסופי שהוא פלט הקול.

תכן מפורט



תיאור האלגוריתמים:

המערכת שפיתחנו נקראת "Read My Lips". המערכת מקבלת כקלט סרטונים אשר מפוצלים לרצף פריימים של תמונות ובהם מתבצע זיהוי הפנים. האלגוריתם מזהה את איזור ה-ROI (range of interest) שהוא אזור השפתיים וחותר אותו עבור כל פריים.

רצף הפריימים המתקבל כקלט עובר דרך שלוש שכבות של רשת נוירונים **STCNN** המבצע חישובי קונבולוציה (3D convolution), שמטרתם לזהות ולקבוע תבניות של תנועות השפתיים מתוך התמונות לאורך זמן.

האלמנטים שחולצו וקודדו לוקטורים מספריים המכילים מידע על רצף הפריימים, מועברות כקלט אל יחידות **GRU's** שהם מסוג **RNN**. מטרתם לסנן בעזרת שערים את המידע מתוך הוקטורים כך שרק המידע הנחוץ יישמר. כך רשת הנוירונים מעבדת את ההברות לרצף תווים הנשמר כמו זיכרון ארוך טווח ע"י יחידות **LSTM**.

לאחר מכן מועבר רצף התווים אל **CTC loss** החוזה את הנאמר ברצף ההברות, מצמצם את כמות ההברות ובכך יוצר את המילים והמשפט כפלט סופי.

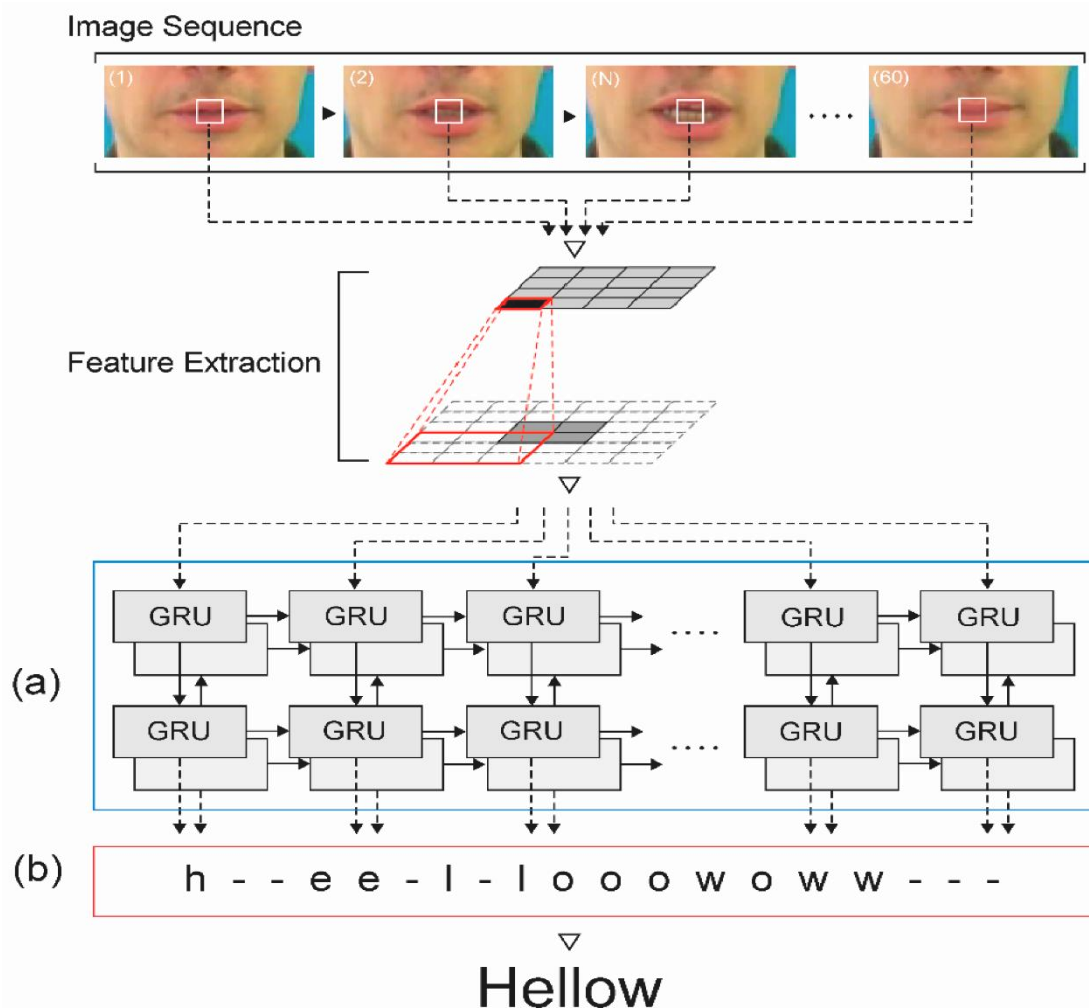
עם שילוב אלגוריתם ה-TTS אנו מקבלים כקלט את חיזוי הטקסט מהאלגוריתם הראשון, קלט זה מפוצל ליחידות קטנות יותר של הברות על מנת לייצר רצף של צלילים המייצגים את הטקסט.

לאחר מכן, המערכת מייצרת אמפליטודה של צלילי השפה ומבצעת התאמה בין הטקסט לקול. בהמשך, המערכת מסנתזת את צורת הדיבור ע"י שינוי של פרמטרים כגון: גובה הצליל, אורכו, וכו', כך שיתאימו למאפיינים הלשוניים של הטקסט.

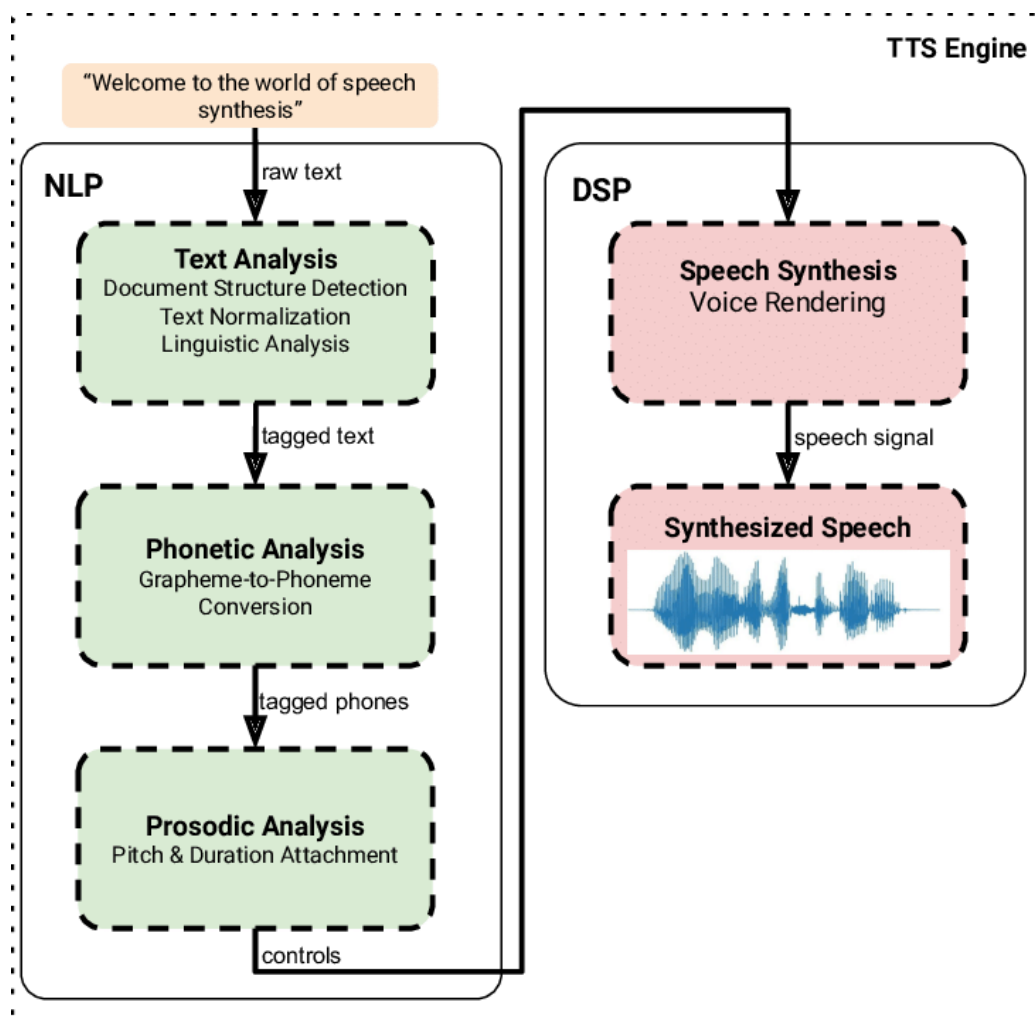
השתמשנו בשרת של IBM ע"י שליחת בקשת API שהוא עוזר במימוש אלגוריתם TTS בכך שמשתמש ב-IBM-WATSON שהוא מערכת מחשב מבוססת בינה מלאכותית.

לבסוף, מערכת ה-TTS מוציאה את פלט הדיבור המסונתז וניתן לשמוע אותו דרך רמקול או לאחסנו בקובץ שמע.

המחשה למשפטיים לטקסט:



המחשה להמרה מטקסט לקול:



תיאור התוצר גרסת ALFHA

אלגוריתמים

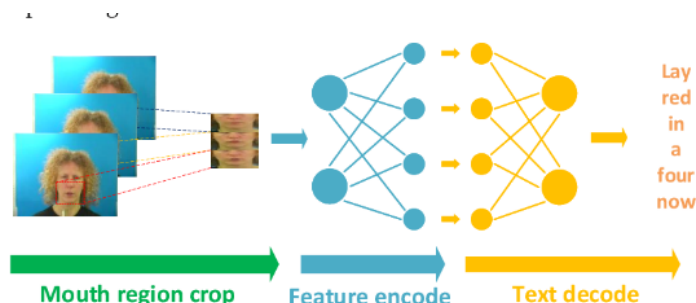
אלגוריתם קריאת השפתיים אשר אחראי לבצע חיזוי של הנאמר מתנועות השפתיים המתבטאת מהם לפלט טקסט. האלגוריתם מקבל כקלט סרטון של דובר אותו הוא מחלק ל-75 פריימים ומהם מחלץ את אזור השפתיים ולאחריו חיזוי הברה אחרי הברה לרצף הברות עד לפלט טקסט סופי.

load_data - מקבלת סרטון ותמלול מקורי תואם ושולחת אותם לפונקציות המעבדות אותם למערכי tensor כהכנה לאימון וחיזוי של המודל ברשת הניורונים. אם בתמלול קיים "sil" הכוונה היא שלא נאמרה שום מילה ויש להמשיך לשאר המילים.

load_video - מקבלת כתובת של מיקום סרטון ומבצעת חילוק שלו לפריימים, לאחר מכן חילוף אזור השפתיים והמרתו למערך tensor המכיל מערך numpy של תתי מערכים המכילים מידע מטיפוס float על הפריימים שבסרטון. לאחר שקיבלנו את מערך הפריימים של השפתיים מבוצע חישוב ממוצע של הפריימים ולאחריו חישוב התפלגות נורמלית ביחס לממוצע וכך קיבלנו מערך preprocessed אשר מכיל מפחית שונות בין צבעים ותאורה בין הפריימים ונשלח לרשת הניורונים.

$$Z_j = \frac{X_j - \text{Mean of } X}{\text{Standard deviation of } X}$$

load_alignments - מקבלת כתובת של התמלול המקורי של הסרטון בהתאם לסרטון שנשלח לחיזוי ומחלקת אותו משפט אחר משפט ומבצע קידוד שלו למערך tensor בעל ערכים נומריים על סמך dictionary של אותיות שהוגדר במערכת. בכך נוכל לבצע קידוד למידע preprocessed ופענוח של המידע לאחר עיבודו ע"י רשת הניורונים.



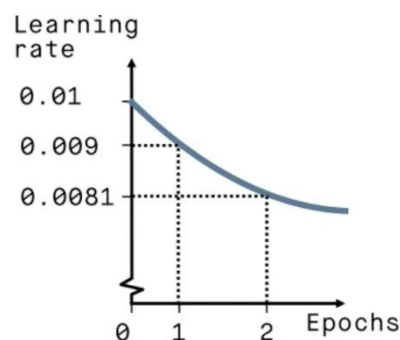
המודל בנוי כך שהוא מורכב מכמה שכבות, כאשר כל שכבה מחשבת חישוביים מתמטיים ומורכבים ומבצעת טרנספורמציה על המערכים המייצגים את הפריימים שלה הסרטונים עד שתצמצם את כמות הפרמטרים ויתקבל חיזוי של רצף הברות ומשם למילים ומשפט סופי.

Layer (type)	Output Shape	Param #
conv3d_18 (Conv3D)	(None, 75, 46, 140, 128)	3584
activation_18 (Activation)	(None, 75, 46, 140, 128)	0
max_pooling3d_18 (MaxPooling3D)	(None, 75, 23, 70, 128)	0
conv3d_19 (Conv3D)	(None, 75, 23, 70, 256)	884992
activation_19 (Activation)	(None, 75, 23, 70, 256)	0
max_pooling3d_19 (MaxPooling3D)	(None, 75, 11, 35, 256)	0
conv3d_20 (Conv3D)	(None, 75, 11, 35, 75)	518475
activation_20 (Activation)	(None, 75, 11, 35, 75)	0
max_pooling3d_20 (MaxPooling3D)	(None, 75, 5, 17, 75)	0
time_distributed_6 (TimeDistributed)	(None, 75, 6375)	0
bidirectional_12 (Bidirectional)	(None, 75, 256)	6660096
dropout_12 (Dropout)	(None, 75, 256)	0
bidirectional_13 (Bidirectional)	(None, 75, 256)	394240
dropout_13 (Dropout)	(None, 75, 256)	0
dense_6 (Dense)	(None, 75, 41)	10537
Total params: 8,471,924		
Trainable params: 8,471,924		

*פירוט שכבות ה-NN של המודל והפרמטרים בכל אחד מהם

Scheduler - פונקציה לחישוב learning rate לאחר כל מחזור epoch באימון המכונה.

ככל שעולה מספר החזרות כך יורד ה-learning rate ע"י הכפלתו במשתנה אקספוננציאלי והמכונה מקבלת פידבק על יכולות הדיוק בחיזוי שלה.



*גרף המתאר את היחס בין learning rate ל-epoch ואת השיפור בכל מחזור של אימון

דוגמת הרצת אימון לאחר 3 מחזורי epoch-

```
model.fit(train, validation_data=test, epochs=100, callbacks=[checkpoint_callback,

Epoch 1/100
1/1 [=====] - 0s 118ms/step loss: 69.06
Original: place blue in b seven soon
Prediction: la e e e eo
-----
Original: place blue by v eight please
Prediction: la e e e eo
-----
450/450 [=====] - 460s 1s/step - loss: 69.0659 - val_loss: 64.3408 - lr: 1.0000e-04
Epoch 2/100
1/1 [=====] - 0s 121ms/step loss: 65.58
Original: lay white with f nine again
Prediction: la e e e eon
-----
Original: set white in u six please
Prediction: la e e e eon
-----
450/450 [=====] - 462s 1s/step - loss: 65.5831 - val_loss: 61.2463 - lr: 1.0000e-04
Epoch 3/100
9/450 [.....] - ETA: 4:33 - loss: 63.1770
```

CTC loss - פונקציה לבדיקת ציון ההתאמה בין החיזוי של הטקסט לתמלול המקורי. ניתן לראות שיפור במהלך האימון של המכונה לכל מחזור של epoch.

ProduceExample Class - מחלקה האחראית על פענוח החיזוי של פונקציית ctc וצמצום רצף ההברות למשפט בעל תחביר הגיוני ומדפיסה את המשפט המקורי בהשוואה למשפט החזוי לאחר כל איטרציה על מספר מסוים של סרטונים. הפלט המתקבל הוא מערך tensor המכיל את פלט הטקסט בצורה הבאה:

```
tf.Tensor: shape=(75, 46, 140, 1), dtype=float32, numpy=
```

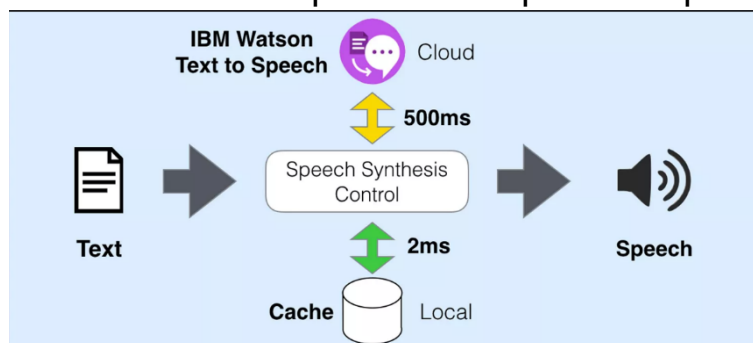
מפלט זה נייצר את המשפט הסופי ע"י decode שאחראי על פענוח הטקסט.

לאחר מכן, נעביר את פלט הטקסט הסופי אל קלט ה-TTS.

אלגוריתם TTS מאפשר המרה של קלט טקסט לקול בכך שהוא מבצע חמישה שלבים עיקריים בהם המשפט מפוצל לתווים ומשם להברות המתוגות ע"י מודל עיבוד שפה (NLP).

בתהליך זה נאספים יחידות ההברה ומועבדים עבורם צלילים התואמים לאורכם ולבסוף נקבל את פלט הקול המלא.

IBM Watson היא חבילה של שירותים וכלים של בינה מלאכותית (AI) ולמידת מכונה (ML) שפותחה על ידי IBM. ווטסון נועדה לספק מגוון פתרונות בינה מלאכותית ביניהם עיבוד שפה טבעית (NLP), למידת מכונה וניתוח נתונים. במערכת שלנו נעשה שימוש במערכת זו ע"י שליחת בקשת API לשרתי IBM המאפשרים לנו קבלת פלט קול מהיר ומדויק מאוד.



קוד

Libraries imports

```
import os
import cv2 #preprocess videos library
import tensorflow as tf #deep learning framework
import numpy as np #preprocess array of data
from typing import List
from matplotlib import pyplot as plt #post process data library
import imageio #convert numpy array to gif
```

פונקציית טעינת סרטונים

```
def load_video(path:str) -> List[float]:
```

```
    """
```

Loads a video from the given path and preprocesses it.

Parameters:

path (str): The path to the video file.

Returns:

List[float]: A list of preprocessed video frames.

This function loads a video from the given path using OpenCV's VideoCapture method and preprocesses each frame by converting it to grayscale and cropping the lip region from the frame. It then calculates the mean and standard deviation of the frames and returns the preprocessed frames as a list of floats.

```
    """
```

```
    cap = cv2.VideoCapture(path)
    frames = []
    #loop over each frame in video
    for _ in range(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))):
        ret, frame = cap.read()
        #convert each frame to grayscale
        frame = tf.image.rgb_to_grayscale(frame)
        #crop of lip region from video
        frames.append(frame[190:236,80:220,:])
    cap.release()

    mean = tf.math.reduce_mean(frames)#calculate mean
    std = tf.math.reduce_std(tf.cast(frames, tf.float32))#calc standerise
    return tf.cast((frames - mean), tf.float32) / std#return list of pre processing data of video
```

פונקציית טעינת טקסט

```
def load_alignments(path:str) -> List[str]:
```

```
"""
```

Loads a file containing alignment information and preprocesses it

Parameters:

path (str): The path to the file containing the alignment information.

Returns:

List[str]: A list of preprocessed alignment tokens.

This function loads a file containing alignment information using Python's built-in file I/O functions. It then preprocesses the file by extracting the alignment tokens from each line, skipping any lines containing silent segments in the video. The tokens are then converted to numerical format and returned as a list of strings.

```
"""
```

```
with open(path, 'r') as f:
    lines = f.readlines()
    tokens = []
    #loop over alignments line by line
    for line in lines:
        line = line.split()
        #if there is silent in video we skip this line
        if line[2] != 'sil':
            tokens = [*tokens, line[2]]
    return char_to_num(tf.reshape(tf.strings.unicode_split(tokens, input_encoding='UTF-8'), (-1,)))[1:]
```

פונקציית טעינת וידיאו והטקסט המתאים לו כמידע מעובד לשליחה לרשת הנוירונים

```
def load_data(path: str):
```

```
"""
```

Loads the video frames and their corresponding alignments for a given file path.

Parameters:

path (str): The file path of the video file.

Returns:

Tuple[List[float], str]: A tuple containing a list of preprocessed video frames and the corresponding alignment string.

This function loads the video frames and their corresponding alignments for a given file path. It first extracts the file name from the file path and uses it to locate the video file and its corresponding alignment file. It then calls the 'load_video' function to preprocess the video frames and the 'load_alignments' function to load the alignment data. The function r

returns a tuple containing the preprocessed video frames and the corresponding alignment string.

```
"""  
  
path = bytes.decode(path.numpy())  
file_name = path.split('/')[-1].split('.')[0]  
# File name splitting for windows  
#file_name = path.split("\\")[-1].split('.')[0]  
video_path = os.path.join('data','s1',f'{file_name}.mpg')  
alignment_path = os.path.join('data','alignments','s1',f'{file_name}.align')  
frames = load_video(video_path)  
alignments = load_alignments(alignment_path)  
  
return frames, alignments
```

יצירת דאטאסט מעובד לרשת הנוירונים והגדרת הפרמטרים המתאימים

```
#create tensorflow data set of videos and alignments and mapping it as tensorflow data  
data = tf.data.Dataset.list_files('./data/s1/*.mpg')  
data = data.shuffle(500, reshuffle_each_iteration=False)  
data = data.map(mappable_function)#mapping dataset  
data = data.padded_batch(2, padded_shapes=([75,None,None,None],[40]))#batch of 2 vi  
deos and 2 alignments  
data = data.prefetch(tf.data.AUTOTUNE)  
# Added for split  
train = data.take(450)#train for first 450 samples  
test = data.skip(450)#test for the rest of samples after the first 450
```

Imports of neural network layers

```
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, MaxPool3D, Activation, Reshape, SpatialDropout3D, BatchNormalization, TimeDistributed, Flatten  
from tensorflow.keras.optimizers import Adam  
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
```

Build neural network architecture

```
model = Sequential()  
model.add(Conv3D(128, 3, input_shape=(75,46,140,1), padding='same'))  
model.add(Activation('relu'))  
model.add(MaxPool3D((1,2,2)))  
  
model.add(Conv3D(256, 3, padding='same'))  
model.add(Activation('relu'))  
model.add(MaxPool3D((1,2,2)))
```

```

model.add(Conv3D(75, 3, padding='same'))
model.add(Activation('relu'))
model.add(MaxPool3D((1,2,2)))

model.add(TimeDistributed(Flatten()))

model.add(Bidirectional(LSTM(128, kernel_initializer='Orthogonal', return_sequences=True)))
model.add(Dropout(.5))

model.add(Bidirectional(LSTM(128, kernel_initializer='Orthogonal', return_sequences=True)))
model.add(Dropout(.5))

model.add(Dense(char_to_num.vocabulary_size()+1, kernel_initializer='he_normal', activation='softmax'))

```

פונקציה המחזירה פידבק ללמידת המכונה אם שיפרה את אחוז הדיוק שלה בחיזוי

```

def scheduler(epoch, lr):
    """

```

Learning rate scheduler function that reduces the learning rate after a certain number of epochs.

Parameters:

epoch (int): The current epoch number.

lr (float): The current learning rate

Returns:

float: The new learning rate.

This function is a simple learning rate scheduler that reduces the learning rate after 30 epochs. It takes the current epoch number and the current learning rate as input parameters and returns a new learning rate. If the current epoch is less than 30, the function returns the current learning rate. Otherwise, it reduces the learning rate by multiplying it with the mathematical constant e raised to the power of -0.1 .

```

    """
    if epoch < 30:
        return lr
    else:
        return lr * tf.math.exp(-0.1)

```

פונקציית חישוב CTCloss - מציאת התאמה בין אותיות לנאמר בסרטון על מנת ליצור רצף הגיוני

```
def CTCLoss(y_true, y_pred):  
    """
```

Computes the Connectionist Temporal Classification (CTC) loss between the true and predicted labels.

Parameters:

y_true (Tensor): The true label tensor.

y_pred (Tensor): The predicted label tensor.

Returns:

Tensor: The CTC loss tensor.

This function computes the CTC loss between the true and predicted labels using the CTC batch cost function from the Keras backend. It takes in the true and predicted label tensors as input parameters and returns a tensor representing the CTC loss. The function also calculates the batch length, input length, and label length, which are required for the CTC batch cost function.

```
    """
```

```
    batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")  
    input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")  
    label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")  
  
    input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")  
    label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")  
  
    loss = tf.keras.backend.ctc_batch_cost(y_true, y_pred, input_length, label_length)  
    return loss
```

מחלקת חיזוי וקבלת פידבק של מכונת הלמידה

```
class ProduceExample(tf.keras.callbacks.Callback):  
    """
```

Callback class that generates examples during training.

Parameters:

dataset (numpy iterator): The dataset iterator.

This class is a callback function that generates examples during training. It takes in a dataset iterator as an input parameter during initialization. The class has an `on_epoch_end` method that is called at the end of each epoch. This method generates predictions for a batch of input data using the model, decodes the predictions using the CTC decoding function, and prints the original and predicted text for each input in the batch.

```
    """
```

```

def __init__(self, dataset) -> None:
    self.dataset = dataset.as_numpy_iterator()

def on_epoch_end(self, epoch, logs=None) -> None:
    data = self.dataset.next()
    yhat = self.model.predict(data[0])
    decoded = tf.keras.backend.ctc_decode(yhat, [75,75], greedy=False)[0][0].numpy()#
decoding ctc model data
    for x in range(len(yhat)):
        print('Original:', tf.strings.reduce_join(num_to_char(data[1][x])).numpy().decode('u
tf-8'))
        print('Prediction:', tf.strings.reduce_join(num_to_char(decoded[x])).numpy().decod
e('utf-8'))
        print('~'*100)

```

הכנה לאימון המכונה

```

#compile model
model.compile(optimizer=Adam(learning_rate=0.0001), loss=CTCLoss)
#save model checkpoints
checkpoint_callback = ModelCheckpoint(os.path.join('models','checkpoint'), monitor='loss
', save_weights_only=True)
#dropdown learning rate for each epoch
schedule_callback = LearningRateScheduler(scheduler)
example_callback = ProduceExample(test)

model.fit(train, validation_data=test, epochs=100, callbacks=[checkpoint_callback, sched
ule_callback, example_callback])

```

חיזוי

```

#load model
model.load_weights('/content/gdrive/MyDrive/LipNet/models/checkpoint')

#iteartor of dataset numpy data
test_data = test.as_numpy_iterator()
sample = test_data.next()

#prediction
yhat = model.predict(sample[0])

#decoding prediciton
decoded = tf.keras.backend.ctc_decode(yhat, input_length=[75,75], greedy=True)[0][0].n
umpy()

```



```

#prints original text
print('~'*100, 'REAL TEXT')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in [sample
[1]]]

#prints the prediction of the machine
print('~'*100, 'PREDICTIONS')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decode
d]

```

פונקציית חישוב CER

```

def calculate_cer(ref,hyp):
    for i in range (len(originalChars)):
        totalCer=totalCer+calculate_wer(ref[i],hyp[i])
    return totalCer

```

פונקציית חישוב WER

```

def calculate_wer(ref, hyp ,debug=False):
    r = ref.split()
    h = hyp.split()
    #costs will holds the costs, like in the Levenshtein distance algorithm
    costs = [[0 for inner in range(len(h)+1)] for outer in range(len(r)+1)]
    # backtrack will hold the operations we've done.
    # so we could later backtrack, like the WER algorithm requires us to.
    backtrack = [[0 for inner in range(len(h)+1)] for outer in range(len(r)+1)]

    OP_OK = 0
    OP_SUB = 1
    OP_INS = 2
    OP_DEL = 3
    DEL_PENALTY = 1
    INS_PENALTY = 1
    SUB_PENALTY = 1

    # First column represents the case where we achieve zero
    # hypothesis words by deleting all reference words.
    for i in range(1, len(r)+1):
        costs[i][0] = DEL_PENALTY*i
        backtrack[i][0] = OP_DEL

    # First row represents the case where we achieve the hypothesis
    # by inserting all hypothesis words into a zero-length reference.

```

```

for j in range(1, len(h) + 1):
    costs[0][j] = INS_PENALTY * j
    backtrace[0][j] = OP_INS

# computation
for i in range(1, len(r)+1):
    for j in range(1, len(h)+1):
        if r[i-1] == h[j-1]:
            costs[i][j] = costs[i-1][j-1]
            backtrace[i][j] = OP_OK
        else:
            substitutionCost = costs[i-1][j-1] + SUB_PENALTY # penalty is always 1
            insertionCost = costs[i][j-1] + INS_PENALTY # penalty is always 1
            deletionCost = costs[i-1][j] + DEL_PENALTY # penalty is always 1

            costs[i][j] = min(substitutionCost, insertionCost, deletionCost)
            if costs[i][j] == substitutionCost:
                backtrace[i][j] = OP_SUB
            elif costs[i][j] == insertionCost:
                backtrace[i][j] = OP_INS
            else:
                backtrace[i][j] = OP_DEL

# back trace though the best route:
i = len(r)
j = len(h)
numSub = 0
numDel = 0
numIns = 0
numCor = 0
if debug:
    # print("OP\tREF\tHYP")
    lines = []
while i > 0 or j > 0:
    if backtrace[i][j] == OP_OK:
        numCor += 1
        i-=1
        j-=1
        if debug:
            lines.append("OK\t" + r[i]+" \t"+h[j])
    elif backtrace[i][j] == OP_SUB:
        numSub +=1
        i-=1
        j-=1
        if debug:
            lines.append("SUB\t" + r[i]+" \t"+h[j])
    elif backtrace[i][j] == OP_INS:
        numIns += 1

```

```

j-=1
if debug:
    lines.append("INS\t" + "*****" + "\t" + h[j])
elif backtrace[i][j] == OP_DEL:
    numDel += 1
    i-=1
    if debug:
        lines.append("DEL\t" + r[j]+"\t"+"*****")
if debug:
    lines = reversed(lines)
    for line in lines:
        print(line)
    print("#cor " + str(numCor))
    print("#sub " + str(numSub))
    print("#del " + str(numDel))
    print("#ins " + str(numIns))
# return (numSub + numDel + numIns)
wer_result = numSub + numDel + numIns
return wer_result

```

הדפסה של תוצאות המדדים לאחר חישוב

```

wer=0
cer=0
lastwer=0
lastcer=0
counterwords=0
counterChars=0
originalChars=[]
predictChars=[]
for i in range(len(originalArray)):
    words=originalArray[i].split() #[set,white,at,blue]
    counterwords=counterwords+len(words)
    wordspredict=predictArray[i].split()
    for j in range (min(len(words), len(wordspredict))):
        cer=cer+calculate_wer(words[j],wordspredict[j])
        counterChars=counterChars+len(words[j])

for index in range(len(originalArray)):
    wer=wer+calculate_wer(originalArray[index],predictArray[index])

lastwer=(int)((wer/counterwords)* 100)
lastcer=(int)((cer/counterChars)* 100)
#presentage of WER
print("-----\n")
print("Measurements:\n")
print("WER:", lastwer,"%", "CER: ",lastcer,"%" )
print("-----\n")

```

המרת מערך הנתונים המתקבל מרשת הנירונים לאחר החיזוי למחרוזת כהכנה לקלט TTS

```
#last contains the last processed sentence after decoding numpy array
last = [tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in d
ecoded]

#convert the array to string as input for tts
t_array = np.array(last)
last_sentence = t_array[0].numpy().decode('utf-8')
print(last_sentence)
```

Imports of TTS machine

```
from ibm_watson import TextToSpeechV1
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator
```

Create authentication to IBM services for text-to-speech

```
apikey = 'jwv3z0rLKVAG4-2MSTOwAQIsnoeA_yubgLwLOaxcurFH'
url = 'https://api.au-syd.text-to-speech.watson.cloud.ibm.com/instances/07d2be72-5a3c-
4da1-85f3-cbab3a60795f'

# Setup Service
authenticator = IAMAuthenticator(apikey)
tts = TextToSpeechV1(authenticator=authenticator)
tts.set_service_url(url)
```

Convert with a basic language model

```
with open('/content/gdrive/MyDrive/TextToSpeechPorject/speech.mp3', 'wb') as audio_fil
e:
    res = tts.synthesize(last_sentence, accept='audio/mp3', voice='en-
US_KevinV3Voice').get_result()
    audio_file.write(res.content)
```

Play the file using lpython library for playing the output sound

```
from IPython.display import Audio
from IPython.display import Audio
sound_file = '/content/gdrive/MyDrive/TextToSpeechPorject/speech.mp3'
wn = Audio(sound_file, autoplay=True)
display(wn)
```

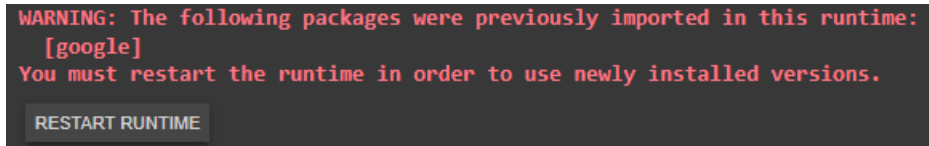
אופן התקנה והרצת הקוד

את קוד זה הרצנו ב-google colab. יש לבצע את ההתקנות הבאות:

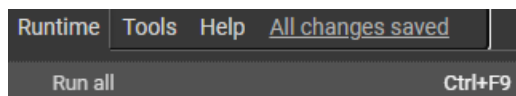
```
!pip install tensorflow==2.10
```

יש לבצע את התקנה זו לפני הרצת התוכנית

לחץ על Restart runtime



לאחר ריסוס זמן הריצה יש ללחוץ על Run All



```
!pip install ibm_watson
```

```
!pip install opencv-python matplotlib imageio gdown
```

לאחר מכן, יש ליצור חיבור לדרייב על מנת להוסיף את הדאטאסט עליו בוצע האימון:

```
from google.colab import drive  
drive.mount('/content/gdrive')
```

הורדת ה-dataset:

```
#download dataset from url link and extratced it to drive  
url = 'https://drive.google.com/uc?id=1YlvpDLix3S-U8fd-gqRwPcWXAXm8JwjL'  
output = 'data.zip'  
gdown.download(url, output, quiet=False)  
gdown.extractall('data.zip')
```

הורדת המודל המאומן:

```
#download checkpoint of trained model  
url = 'https://drive.google.com/uc?id=1vWscXs4Vt0a_1IH1-ct2TCgXAZT-N3_Y'  
output = 'checkpoints.zip'  
gdown.download(url, output, quiet=False)  
gdown.extractall('checkpoints.zip', 'models')
```

טעינת המודל:

```
model.load_weights('/content/gdrive/MyDrive/LipNet/models/checkpoint')
```

בסופה של הרצת התוכנית ייבחר סרטון אקראי להדגמת ריצת התוכנית

טעינת סרטון לחיזוי:

לאחר כל השלבים הללו ניתן לחזות את הנאמר מכל סרטון שנרצה, בשילוב האלגוריתם השני נקבל גם פלט קול

```
#file path of my own video
filePath='/content/gdrive/MyDrive/LipNet/data/s1/swab6n1.mpg'

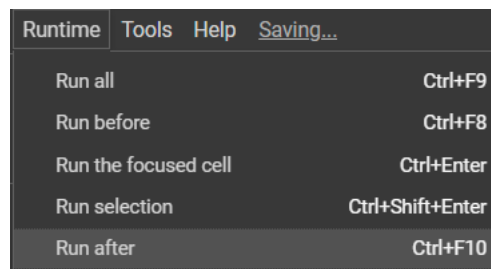
sample = load_data(tf.convert_to_tensor(filePath))
```

הדגמה

כאשר נרצה לבצע חיזוי יש לטעון את הסרטון לשורה הבאה:

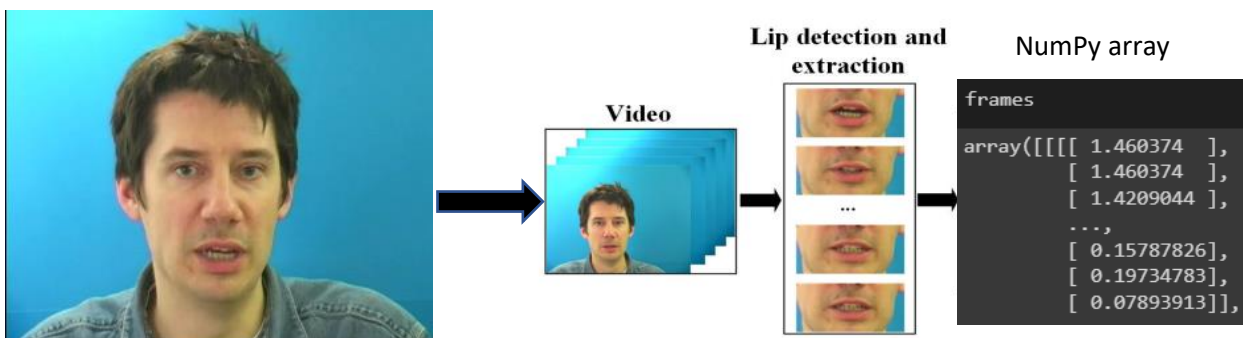
```
#file path of my own video  
filePath='/content/gdrive/MyDrive/LipNet/data/s1/pgbk8p.mpg'
```

לאחר מכן נלחץ על Runtime->Run after על מנת שלא להריץ מחדש את הפרויקט אלא רק מנקודה זאת.



הסרטון יישלח אל פונקציית `load_data` שלמעשה מכינה את הסרטון למידע מעובד במערך `NumPy` בעל ערכים נומריים כהכנה לרשת הנוירונים.

משם יישלח הסרטון אל פונקציית `load_video` המפצלת את הסרטון ל-75 פריימים, מחלצת את אזור השפתיים מכל פריים ומקודדת את המידע שלו.



השימוש במערך `NumPy` מאפשר לנו ריצה מהירה על מידע נומרי והוא לא זול זיכרון רב מהמערכת. כמו כן, השימוש בו מאוד פופולרי בלמידת מכונה כאשר יש צורך בחישובים מתמטיים ולוגיים מורכבים על כמות מידע גדולה.

לאחר שקיבלנו את מערך הפריימים כמערך `preprocessed` מתבצעת המרה למערך מסוג `tensor` שהוא ספריית פייתון של למידת מכונה של גוגל, שאליו מתווספים אלמנטים נוספים של הסרטון:

```
tf.Tensor: shape=(75, 46, 140, 1), dtype=float32, numpy=
```

אחד מהמשתנים של מערך זה הוא `shape` והוא מגדיר את מבנה המערך בעת שליחתו לרשת הנוירונים, במערכת שלנו כל סרטון מחולק ל-75 פריימים המיוצגים במערך בתור 75 תתי מערכים בעלי רזולוציה של 46×140 פיקסלים ובעל ערוץ יחיד שהוא אפור.

הפריימים מומרים לצבע אפור כיוון שכך עיבוד המידע מהיר יותר ודורש פחות זמן ריצה מאשר תמונה שהיא צבעונית.

לאחר שהמידע על הפריימים מעובד הוא נשלח אל רשת הנוירונים.

```
yhat = model.predict(tf.expand_dims(sample[0], axis=0))
```

*שורת שליחה של מידע `preprocessed` אל רשת הנוירונים.

לשם ההדגמה, ניתן לראות חלק מתהליך החיזוי בעת שהמערכת מפענחת את המידע הויזואלי לרצף הפריימים:



המידע המתקבל מרשת הנוירונים מפוענח ע"י פונקציית ה-`ctc_decode` שמטרתה פענוח הפלט הנומרי לתמלול משמעותי. זה כרוך בהסרת תווים כפולים, הסרת רווחים, וצמצום תווים חוזרים כדי לייצר תמליל סופי.

```
decoded = tf.keras.backend.ctc_decode(yhat, input_length=[75], greedy=True)[0][0].numpy()
```

לאחר חיזוי של המודל נקבל את הטקסט החזוי שלנו:

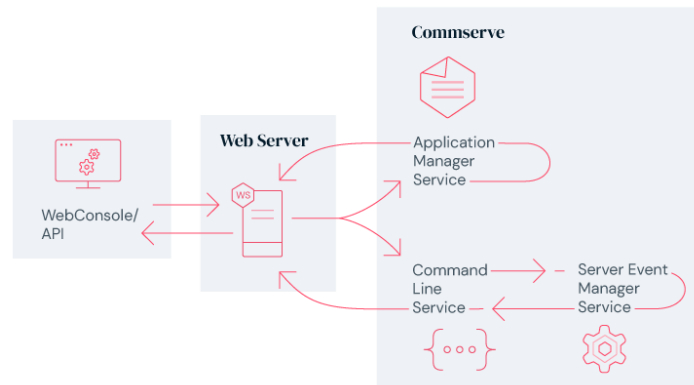
```
<tf.Tensor: shape=(), dtype=string, numpy=b'place green by eight please'>
```

פלט הטקסט מתקבל כמערך `Tensor` ועל מנת שנוכל לשלבו עם מערכת TTS ביצענו חילוץ ופענוח של הטקסט שיהווה קלט עבור `Text-To-Speech`.

```
t_array = np.array(last)
last_sentence = t_array[0].numpy().decode('utf-8')
print(last_sentence)

place green by eight please
```


מערכת ה-TTS עושה שימוש בשרתי IBM ולכן על מנת שנוכל לגשת לשירותים הללו עלינו לייצר access token כך שתתאפשר הגישה אליו.



Access token הוא מפתח מוצפן אשר בעזרתו אנו ניגשים לשרת ועושים שימוש במערכת ה-TTS.

```
apikey = 'jwv3z0rLKVAG4-2MST0wAQ1snoeA_yubgLwL0axcurFH'  
url = 'https://api.au-syd.text-to-speech.watson.cloud.ibm.com/instances/07d2be72-5a3c-4da1-85f3-cbab3a60795f'
```

לאחר שליחת הבקשה נקבל את פלט הקול לקובץ שמע:

```
from IPython.display import Audio  
from IPython.display import display  
sound_file = '/content/gdrive/MyDrive/LipNet/audio/speech'+ '_' + filename + '.wav'  
wn = Audio(sound_file, autoplay=True)  
display(wn)
```

הערכה

DATA SET

ה-data set שבו השתמשנו על מנת לאמן את המודל שלנו ולעשות וולידציה עליו הוא: **Dataset GRID** שמכיל כ- 34 דוברים שכל אחד מהם מייצר כ-1000 משפטים ובסך הכל 34,000 משפטים. הצילום נעשה לאורך 28 שעות סה"כ.

ה-data set מורכב משני חלקים, הראשון הוא אוסף סרטונים והשני זה הטקסט המתאים לכל סרטון. כך למעשה אנו מאמנים את המערכת להבין שפה וויזואלית.

עשינו מניפולציה על ה-dataset בכך שצמצמנו ממדים, צמצמנו את צילום הדובר לאזור העניין שהוא אזור הפה ובכך התמקדנו בשפתיו של המצולם זאת בשביל להבחין בשינוי תנועת שפתיו בעת הדיבור ובכך האלגוריתם שלנו יזהה את ההברות המתאימות של הדובר (משם את האותיות, חיבור האותיות למילים ומשם למשפטים).

חלוקת ה-dataset: TRAINING - כ-80 אחוז מהסרטונים, TESTING - 20 אחוז הנותרים, VALIDATION - כ-5 אחוז מתוך ה-training.


Overall corpus description

- 34 Speakers (18 male, 16 female)
- 1000 sentences each
- 25 fps video data
- Sentences are of the form:
 - "put red at G9 now"
- Alignment available

Provided Labeling

```
0 15500 sil
15500 22250 bin
22250 28000 red
28000 31000 with
31000 35000 g
35000 44250 seven
44250 51750 soon
51750 74500 sil
```

Screenshots of speakers



מדדים

לשם בדיקת אחוז הדיוק של המערכת נעשה שימוש בשתי מדדים:

1. **CER (character error rate)** - אחוז השגיאות בחיזוי האותיות במילה.

$$CER = \frac{S + D + I}{N}$$

2. **WER (word error rate)** - אחוז השגיאות בחיזוי המילים במשפט.

$$WER = \frac{S_w + D_w + I_w}{N_w}$$

אלו עוזרים לנו לבדוק האם פלט הטקסט תואם את הטקסט המקורי בסרטון. ככל שנבצע יותר סבבי אימונים (epochs) בסביבות ה-30 מחזורים נראה שיפור משמעותי בחיזוי.

הנוסחאות מורכבות מכמה משתנים:

- **S** = Number of **S**ubstitutions
- **D** = Number of **D**eletions
- **I** = Number of **I**nsertions
- **N** = Number of characters/words in reference text (aka ground truth)

צורת הבדיקה

בדקנו את אלגוריתם שלנו על 500 סרטונים מתוך 1000, אותם חילקנו כך:

450- סרטונים כ- TRAIN (80%).

50- סרטונים כ- TEST (20%).

ביצענו השוואות בין המשפט המקורי למשפט אותו חזינו בהתאמה ובדקנו מה אחוז השגיאות במילים (WER) ומה אחוז השגיאות באותיות (CER) על כל התוצאות של 50 הסרטונים אותם הרצנו ב-TEST, ובכך ראינו את אחוז הדיוק של כל המשפטים שחזינו מכל הסרטונים.

צילום המבדקים שלנו:

פונקציית החישוב של WER ו-CER:

```
def calculate_wer(ref, hyp, debug=False):
    r = ref.split()
    h = hyp.split()
    # costs will holds the costs, like in the Levenshtein distance algorithm
    costs = [[0 for inner in range(len(h)+1)] for outer in range(len(r)+1)]
    # backtrace will hold the operations we've done.
    # so we could later backtrace, like the WER algorithm requires us to.
    backtrace = [[0 for inner in range(len(h)+1)] for outer in range(len(r)+1)]

    OP_OK = 0
    OP_SUB = 1
    OP_INS = 2
    OP_DEL = 3
    DEL_PENALTY = 1
    INS_PENALTY = 1
    SUB_PENALTY = 1

    # First column represents the case where we achieve zero
    # hypothesis words by deleting all reference words.
    for i in range(1, len(r)+1):
        costs[i][0] = DEL_PENALTY*i
        backtrace[i][0] = OP_DEL

    # First row represents the case where we achieve the hypothesis
    # by inserting all hypothesis words into a zero-length reference.
    for j in range(1, len(h) + 1):
        costs[0][j] = INS_PENALTY * j
        backtrace[0][j] = OP_INS

    # computation
    for i in range(1, len(r)+1):
        for j in range(1, len(h)+1):
            if r[i-1] == h[j-1]:
                costs[i][j] = costs[i-1][j-1]
                backtrace[i][j] = OP_OK
            else:
                substitutionCost = costs[i-1][j-1] + SUB_PENALTY # penalty is always 1
                insertionCost = costs[i][j-1] + INS_PENALTY # penalty is always 1
                deletionCost = costs[i-1][j] + DEL_PENALTY # penalty is always 1

                costs[i][j] = min(substitutionCost, insertionCost, deletionCost)
                if costs[i][j] == substitutionCost:
                    backtrace[i][j] = OP_SUB
                elif costs[i][j] == insertionCost:
                    backtrace[i][j] = OP_INS
                else:
                    backtrace[i][j] = OP_DEL

    wer_result = numSub + numDel + numIns
    return wer_result
```

*החישוב ואח"כ הדרך למציאת המסלול הטוב ביותר ולבסוף החזרה של תוצאת החישוב.

קוד הקריאות לפונקציות החישוב והדפסת הפלט שלהם:

```
wer=0
cer=0
lastwer=0
lastcer=0
counterwords=0
counterChars=0
originalChars=[]
predictChars=[]
for i in range(len(originalArray)):
    words=originalArray[i].split() #[set,white,at,blue]
    counterwords=counterwords+len(words)
    wordspredict=predictArray[i].split()
    for j in range (min(len(words), len(wordspredict))):
        cer=cer+calculate_wer(words[j],wordspredict[j])
        counterChars=counterChars+len(words[j])

for index in range(len(originalArray)):
    wer=wer+calculate_wer(originalArray[index],predictArray[index])

lastwer=(int)((wer/counterwords)* 100)
lastcer=(int)((cer/counterChars)* 100)
#percentage of WER
print("-----\n")
print("Measurements:\n")
print("WER:", lastwer,"%", "CER: ",lastcer,"%" )
print("-----\n")
```

הפלט הסופי:

Measurements:

WER: 12 % CER: 7 %

תוצאות

בפרויקט שלנו ישנם כמה מדדים שאפשרו לנו לקבל מידע על אחוז דיוק המערכת והתוצאות שלה:

Character error rate - אחוז שגיאות אותיות במשפט נתן לנו אינדיקציה כמה אותיות מתוך פלט המשפט הסופי היו שגויות בהשוואה לתמלול המקורי של הסרטון שנבדק. עבור מבדק זה ביצענו חיזוי של 50 סרטונים וקיבלנו אחוז שגיאות של 7% שהוא אחוז יחסית נמוך בהשוואה לאלגוריתמים אחרים העוסקים בקריאת שפתיים אך עדיין רחוק במקצת מתוצאות המאמר שעמדו על 6.4%

Word error rate - אחוז שגיאות המילים במשפט נתן לנו אינדיקציה על מספר המילים השגויות שהתקבלו בפלט המשפט הסופי לאחר חיזוי בהשוואה למילים במשפט המקורי. גם עבור מבדק זה ביצענו חיזוי על אותו מספר סרטונים שהיו חלק ה-test של ה-dataset. תוצאות אחוז השגיאות במבחן זה עמד על 12% כאשר במאמר אחוז השגיאות עמד על 11.4%.

Measurements:

WER: 12 % CER: 7 %

דוגמא לחיזוי של טקסט לפני אימון בהשוואה לחיזוי של המודל המאומן:

לפני אימון:

```
<tf.Tensor: shape=(), dtype=string, numpy=b'77777777333iiiiiiiiiiiiiiiiiaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'>
```

אחרי אימון המכונה:

```
<tf.Tensor: shape=(), dtype=string, numpy=b'set blue with five soon'>
```

בשילוב אלגוריתם ה-TTS במערכת קיבלנו אחוז דיוק של 95% בהמרה מטקסט לקול.

לסיכום, ניתן לראות כי חיזוי המערכת הינו הצלחה ועם אימון נוסף על עוד נתונים מגוונים ניתן להגיע לדיוק גבוה יותר.

סיכום ומסקנות

מטרתה של המערכת שלנו היא פענוח תנועות שפתיים והמרתו לטקסט ומשם לקול כך שנוכל להנגיש ולהקל על אוכלוסיית החירשים, כבדי השמיעה והאילמים בהבנת תקשורת בין אנשים ולגשר על פערי הלקות עם אוכלוסיית השומעים.

מהתוצאות שלנו אנו רואים שאכן מטרתנו הושגה בהצלחה רבה עם אחוז שגיאות קטן ובהחלט המערכת תוכל להקל על אוכלוסיית החירשים, כבדי השמיעה והאילמים בהבנת תקשורת בין אנשים.

היו כמה בעיות בדרך שהצלחנו להתגבר עליהם כגון:

-החיבור בין שני האלגוריתמים, אלגוריתם קריאת שפתיים לטקסט ואלגוריתם טקסט לקול.

-טעינת הקבצים המתאימים לפרויקט והרצה שלו שלקח לנו זמן רב עם שגיאות רבות שתוקנו.

-התמודדנו עם שפת פייתון שלא למדנו לפני כן, ולמדנו אותה תוך כדי תנועה בפרויקט.

-התמודדנו עם פרויקט של "machine learning" שלא למדנו את הנושא הזה לפני כן ואכן זה היה מאוד מאתגר לקפוץ למים העמוקים בפרויקט מורכב כמו שלנו.

אפשרויות שיפור והמלצות להמשך בפרויקט שלנו:

-ניתן לשפר יותר את התוצאות ולהקטין את אחוז השגיאות שהיו לנו.

-אפשר לממשק את הפרויקט שלנו עם מערכות כמו: משקפיים חכמים שיציגו בזמן אמת את הטקסט של מי שמדבר מולך, אפליקציות שיציגו את הטקסט של מי שמדבר מולך וישמיעו את הקול של אדם אילם שמדבר מולך, ועוד מגוון פלטפורמות שיכולות לעזור לבעלי הלקויות.

הערות אישיות:

נהנינו מאוד לעבוד על הפרויקט המאתגר הזה אשר חשף אותנו לעולמות תוכן מגוונים שלא הכרנו לפני כן עם הבנה מעמיקה יותר בפיתוח מערכת "machine learning" בשפת פייתון שזה תחום שרק הולך ומתפתח בעולמינו. נהנינו לעבוד בעבודת צוות מגובשת ויצירתית אחד עם השני ועם סיעור מוחות והצעת רעיונות ששיפרו את עבודתנו המון.

רשימת מקורות

המאמרים שבהם השתמשנו לסקירת הספרות: (References)

[1]- Lip Reading to Text using Artificial Intelligence

Dr. Mamatha G1

Head of the Department

Information Science & Engineering

Nagarjuna college of Engineering & Technology

Bangalore, India

[2] LIPNET: END-TO-END SENTENCE-LEVEL LIPREADING

Yannis M. Assael^{1,†}, Brendan Shillingford^{1,†}, Shimon Whiteson¹

& Nando de Freitas^{1,2,3} Department of Computer Science,

University of Oxford, Oxford, UK ¹ Google DeepMind, London, UK

² CIFAR, Canada ³

[3]- Diverse Pose Lip-Reading Framework

Naheed Akhter ¹, Mushtaq Ali ¹, Lal Hussain ^{2,3}, Mohsin Shah ⁴,

Toqeer Mahmood ⁵ and Amjad Ali ⁶

and Ala Al-Fuqaha ^{6,*}

[4] - Deep Voice: Real-time Neural Text-to-Speech

Sercan O' . Arık * ¹ Mike Chrzanowski * ¹ Adam Coates * ¹

Gregory Diamos * ¹ Andrew Gibiansky * ¹

Yongguo Kang * ² Xian Li * ² John Miller * ¹ Andrew Ng * ¹

Jonathan Raiman * ¹ Shubho Sengupta * ¹

Mohammad Shoeybi * ¹

עזרים נוספים: (References)

-LIP READING: Machine Learning Project at DSR

https://www.youtube.com/watch?v=xkf8kibDntE&t=337s&ab_channel=DataScienceRetreat

-Evaluate OCR Output Quality with Character Error Rate (CER) and Word Error Rate (WER)

<https://towardsdatascience.com/evaluating-ocr-output-quality-with-character-error-rate-cer-and-word-error-rate-wer-853175297510>

-Grid documentary and dataset

<http://spandh.dcs.shef.ac.uk/gridcorpus/>

-IBM language models

<https://cloud.ibm.com/docs/speech-to-text?topic=speech-to-text-models>

-Relu activation function

https://www.youtube.com/watch?v=68BZ5f7P94E&ab_channel=StatQuestwithJoshStarmer

-Activation Functions explanation

https://www.youtube.com/watch?v=s-V7gKrsels&ab_channel=CodeEmporium