**סיווג פסולת באמצעות למידת מכונה**

**פרויקט סופי מדעי המחשב**

**שם התלמיד:** אורי ברכה

**שם בית ספר:**  תיכונט

**תעודת זהות:** 325742682

**שם המנחה**: גד לידרור

**שם החלופה:** למידת מכונה

**תאריך הגשה:**



# תוכן עניינים

תוכן

[תוכן עניינים 1](#_Toc98688353)

[מבוא 2](#_Toc98688354)

[הסבר על מסד נתונים 3](#_Toc98688355)

[YOLO ומסדי נתונים 3](#_Toc98688356)

[הסבר על מסד הנתונים TACO 3](#_Toc98688357)

[שינוי התוויות של TACO 3](#_Toc98688358)

[עריכת קובץ ה- annotation 4](#_Toc98688359)

[שינוי גודל התמונה וגודל ה- annotation 5](#_Toc98688360)

[שינוי ה- annotation מפורמט COCO לפורמט YOLO 5](#_Toc98688361)

[**מסד נתונים שני: מסד נתונים נבנה אישית** 8](#_Toc98688362)

[הסבר על תוכנות תיוג 8](#_Toc98688363)

[בניית מסד הנתונים 9](#_Toc98688365)

[הסבר על **YOLO** 13](#_Toc98688366)

[אימון וחקירת **HyperParameters** 16](#_Toc98688367)

[הקדמה על האימון 16](#_Toc98688368)

[הסבר על האימון 16](#_Toc98688369)

[אימונים 17](#_Toc98688370)

[אימון ראשון 17](#_Toc98688371)

[אימון שני 18](#_Toc98688372)

[אימון שלישי 19](#_Toc98688373)

[אימון רביעי 20](#_Toc98688374)

[אימון חמישי 21](#_Toc98688375)

[אימון שישי 22](#_Toc98688376)

[אימון שביעי 23](#_Toc98688377)

[אימון שמיני 23](#_Toc98688378)

[אימון תשיעי 24](#_Toc98688379)

[סיכום האימונים 26](#_Toc98688380)

[תיאור יצירת היישום 27](#_Toc98688382)

[מדריך למשתמש 34](#_Toc98688383)

[רפלקציה 35](#_Toc98688384)

[ביבליוגרפיה 37](#_Toc98688385)

# מבוא

כמות הפסולת הנזרקת כיום היא בעיה משמעותית מאוד. יש לה השפעה רחבה על בעיית שינוי האקלים שמספר פעמים נקראה הבעיה הכי משמעותית במאה ה- 21. בישראל בלבד יוצרו 13 מיליון טונות של פסולת ב- 2019. אחת הדרכים להפחית את השפעת הזיהום הנגרם מפסולת היא מיחזור.

לפי ויקיפדיה מיחזור הוא תהליך שבו מפרקים פסולת למצב של חומר גלם ומשתמשים בחומרי הגלם כדי ליצור מוצרים חדשים. כתוצאה מכך המיחזור מפחית את הצורך בייצור מוצרים חדשים ולכן מפחית זיהום אוויר ושינוי אקלים.

בישראל (וברוב העולם) המיחזור נעשה באמצעות פחי מיחזור. בעולם קיימים שישה סוגים של פחי מיחזור: מתכת, פלסטיק, נייר, זכוכית, קרטון (אריזות) ופסולת אורגנית. גם בישראל הם קיימים, אך לא כולם נמצאים באזורים קרובים מספיק למקומות דיור באופן שיוכלו להשתמש בהם באופן סדיר. הפחים נלקחים למתקנים מיוחדים בהם מפרידים שוב את הפסולת על מנת לוודא שלא נזרקה לפח הלא נכון ורק אז מבצעים את תהליך המיחזור.

בחרתי את הנושא מכיוון שמדובר בנושא משמעותי, אשר מאוד מעניין אותי ובעבר אף ביצעתי עבודה דומה של למידת מכונה בנושא זה.

מטרתי בפרויקט היא ליצור אפליקציה שיכולה לנתח תמונה שיש בה מספר פריטי פסולת באופן שתסמן כל פריט פסולת הנמצא בתמונה ותציין לאיזה פח הוא מתאים. קהל היעד בפרויקט זה הוא הציבור הרחב ובעיקר הציבור המעוניין למחזר ולא בטוח לאיזה פח לזרוק את הפסולת שהוא מייצר.

למיטב ידיעתי אין אפליקציה המבצעת בדיוק את מה שאני מעוניין לעשות. יש לא מעט אפליקציות שנותנות מידע על נושא המיחזור, מיקומים של פחי מיחזור ואפילו מאפשרות להזמין משאית לסילוק הזבל של הבניין. מצאתי גם אפליקציה בשם Junker, המאפשרת לסרוק פריט אחד ולקבל הסבר לאן יש לזרוק אותו, אך אין שום אפליקציה שעובדת על מספר חפצים בו זמנית.

למטרה זו עלי להשתמש בסוג מסויים של מודלים בלמידת מכונה שנקרא Object Detection. בתחום זה יש שתי ארכטיקטורות עיקריות - (SSD (Single Shot Detector ו- (YOLO (You Only Look Once ולכל אחת מהן יתרונות וחסרונות. היתרון של SSD הוא היותו מהיר יותר בזיהוי התמונות עצמן, מכיוון שהרשת שלו פחות גדולה. החיסרון של SSD הוא העובדה שיש לו בעיה לזהות חפצים קטנים. לעומת זאת, YOLO מתאמן מהר יותר ויכול לזהות חפצים קטנים יותר טוב.

בפרויקט יש צורך לזהות חפצים קטנים בתוך התמונה ולכן הפתרון המתבקש הוא שימוש במודל של YOLO. הסבר מורחב על YOLO יבוא בהמשך.

בפרויקט היו לי שני אתגרים מרכזיים - הראשון הוא למצוא מסד נתונים מתאים ובעיקר מסד נתונים שבו יש את הנתונים בפורמט ש- YOLO דורש. האתגר השני הוא לימוד ה- YOLO. הסבר על מסד הנתונים מופיע בהמשך.

# הסבר על מסד נתונים

## YOLO ומסדי נתונים

מודל YOLO ומודלי Classification עובדים כך שהם לוקחים מסד נתונים, במקרה שלי תמונות מתויגות ולומדים מהן איך לזהות תמונות חדשות שלא אומנו עליהן. מסיבה זו חלק משמעותי מהעבודה הוא מציאת ועיבוד מסד נתונים שיתאים למטרות העבודה.

בעבודה זו השתמשתי בשני מסדי נתונים, הראשון שניסיתי להשתמש בו הוא מסד נתונים ששמו TACO (ראשי תיבות של Trash Annotations in Context). מסד נתונים זה עשה בעיות ולכן בניתי מסד נתונים בעצמי. הסבר על כל אחד ממסדי הנתונים יבוא בתת-פרק נפרד.

## הסבר על מסד הנתונים TACO

מסד הנתונים TACO נאסף מעבודה של אנשים פרטיים ויש בו שני חלקים. החלק הראשון הוא 1,500 תמונות של פסולת בכל מיני תנאים והחלק השני הוא annotation של bounding boxes לכל אחת מהתמונות שמסמנות את מיקום הפסולת בתמונה.

המידע הזה מאפשר, לכאורה, שימוש במסד הנתונים זה לאימון מודל YOLO יחסית בקלות. עם זאת היה צורך בביצוע הרבה preprocessing לפני השימוש במסד הנתונים זה בשביל YOLO.

ה- preprocessing כלל ארבעה שינויים עיקריים:

* שינוי הקטגוריות של ה- annotation ב- TACO לקטגוריות הרצויות
* עריכת קובץ ה- annotation בשביל שיהיה יותר קריא (מחיקת מידע לא נחוץ)
* שינוי גודל התמונות לגודל אחיד
* שינוי גודל ה- annotation בשביל שיתאימו לגודל התמונה

### שינוי התוויות של TACO

ה- annotations של TACO ניתנו בפורמט ששמו COCO. בפורמט COCO הקטגוריות מתחלקות לקטגוריות וקטגוריות-על. החלטתי לשנות את קטגוריות-העל ולהשתמש בהן בשביל התוויות ל- YOLO.

בקובץ ה- annotation כל קטגוריה נראתה כך:

{catgory\_id:id,catgory\_name:name,super\_catgory:super\_catgory}

וכל קטגורית-על נראית כך:

{super\_catgory\_id:id,name:name}

בקובץ כל אחת מהן מיוצגת בליסט נפרד וביחד בליסט מאוחד ולכן בקוד שכתבתי בצעתי את הדברים הבאים:

* טענתי את קובץ ה- JSON שמכיל את המידע של ה- annotation לשלושה ליסטים שונים: אחד לתמונות, אחד לסימנים ואחד לקטגוריות וקטגוריות-על
* יצרתי שבע רשימות שונות (לכל אחת מהתוויות שאני רוצה). כל אחת מהן מכילה את המספר הסידורי של קטגוריות-העל המתאימות לאותה תווית
* עברתי על החלק של קטגוריות-העל באמצעות for loop
* בתוך הלולאה בדקתי בכל קטגורית-על לאיזו תווית היא שייכת (בשימוש במספר הסידורי שלה) ושיניתי את השם שלה לאותה תווית
* עברתי על כל אחת מהקטגוריות הרגילות באמצעות for loop ושיניתי את קטגורית-העל שלה לפי הקטגוריה שהיתה לה קודם
* בסוף כל הקוד שמרתי את המידע החדש בקובץ JSON בשם annotation\_mine.json על מנת שיהיה לי גיבוי של הקובץ הקודם במקרה שמשהו ישתבש

### עריכת קובץ ה- annotation

לפני שינוי גודל התמונות לגודל קבוע, רציתי שקובץ ה- annotation יהיה יותר קומפקטי וקל לקריאה על מנת שלא אעשה טעויות.

הקובץ היה מורכב מ-4 חלקים:

* מידע: שנת ייצור הקובץ, מי ייצר אותו וכד'. את המידע הזה יכולתי להוריד בבטחה
* תמונות: מיקום (file path), אורך ורוחב כל תמונה, באיזה כתובת אפשר למצוא אותה מבחינת URL ומתי הוספה. מידע זה יכולתי להוריד בבטחה
* annotation: כל annotation הכיל מידע על segmentation, שמשתמשים בו לסוג מודל אחר, bounding boxes, שהתחלקו למיקום התיבות בתמונה ושטח התיבות (השטח לא היה הכרחי), קטגוריות ה- annotation ומי הוסיף אותן (מידע לגבי ההוספה לא היה הכרחי)
* קטגוריות: הקטגוריות התחלקו לשניים, קטגוריות רקע, המתארות את הרקע של התמונות (שלא היו הכרחיות) וקטגוריות של החפצים עצמם (שהיו הכרחיות)

בסך הכל נזקקתי למידע הבא: לגבי התמונות - מיקום קובץ, גודל ורוחב, לגבי ה- annotation - מספר סידורי, bounding box וקטגוריה ולגבי הקטגוריות - מידע על קטגוריות החפצים וקטגוריות על.

למטרה זו כתבתי קוד שעושה את הדברים הבאים:

* טוען את הקובץ של ה- annotation שיצרתי בקוד הקודם (annotation\_mine.json) לשלושה ליסטים: אחד בשביל התמונות, השני בשביל ה- annotation והשלישי בשביל הקטגוריות
* מכניס את החלקים שאני צריך למשתנים נפרדים ומוסיף אותם ל- dictonary
* משתמש בספריית JSON בשביל ליצור קובץ חדש בשם annotation\_mine\_short.json

### שינוי גודל התמונה וגודל ה- annotation

התמונות של TACO הן בגדלים שונים, הגדולה ביותר שמצאתי היתה בגודל 5000\*4000 פיקסלים. בשביל לאמן מודל YOLO אני צריך תמונות בגודל אחיד ובגודל מתחלק ב- 32 ללא שארית ולכן היה צורך לשנות את גודל התמונות.

YOLO אומן על מסד נתונים עם תמונות בגודל 416\*416, אבל בחרתי להשתמש בגודל 256\*256 משתי שתי סיבות עיקריות - הראשונה היא בעיית הזיכרון: לא הייתי בטוח כמה מקום יש ל- Google Colab ב- GPU ומאחר ויש לי 1,500 תמונות לא רציתי להסתכן בשגיאת Out Of Memory.

הסיבה השניה היא מהירות האימון: ככל שהתמונות גדולות יותר יקח יותר זמן לאמן את המודל. בנוסף, YOLO היא רשת גדולה עם הרבה פרמטרים והרבה שכבות ולכן העדפתי לספק תמונות קטנות יותר ולא להקשות על אימון המכונה מבחינת RAM ומשאבים אחרים, בעיקר כי רציתי לעשות מספר סבבי אימונים.

בנוסף לשינוי גודל התמונות היה צורך לשנות את ה- annotation בכדי שיתאימו לתמונות החדשות, ליתר דיוק - להכפיל כל אחד מהגדלים של ה- annotation במספר שנוצר מהמשוואה הבאה:

רוחב: 256 / רוחב ישן של תמונה (בשביל קואורדינטות x של annotation)

אורך: 256 / אורך ישן של תמונה (בשביל קואורדינטות y של annotation)

למטרה זו כתבתי קוד שמבצע את הדברים הבאים:

* טוען את התמונות ואת ה- annotation למערכים נפרדים
* יוצר מערך numpy בצורה של (1500,2) שיכיל את המספרים שיוצאים מהמשוואה שכתבתי למעלה
* מכניס את התוצאה של המשוואה למעלה לכל תמונה

לכל תמונה יש שני ערכים - רוחב ואורך. הרוחב ממוקם במקום הראשון (image\_id,0) והאורך במקום השני (image\_id,1)

* משתמש בספריית PIL כדי לפתוח ולשנות את גודל התמונות ל- 256\*256 ומשנה את הערך של האורך והרוחב ל- 256 במערך של התמונות
* שומר את כל התמונות בתיקיה חדשה ששמה images, נותן לכל אחד מהם שם על סמך מספר סידורי ומשנה את הערך של מיקום הקובץ במערך של התמונות
* שומר את המידע החדש בקובץ JSON בשם annotation\_resized.json.

לאחר שהרצתי את כל הקוד, המידע היה כמעט מוכן לשימוש וכל מה שנותר לעשות הוא להפוך את ה- annotation מפורמט COCO לפורמט YOLO.

### שינוי ה- annotation מפורמט COCO לפורמט YOLO

COCO (ראשי תיבות של Common Objects in Context), הוא מסד נתונים מאוד פופולרי, שמשתמשים בו בתחרויות של זיהוי אובייקטים בכל העולם (אפילו הגרסאות של YOLO נבדקות עליו). לכן מאגרי מידע אחרים נוצרים בפורמט דומה בכדי שיהיה קל יותר להשתמש בהם. מסיבה זו התחלתי לחפש מאגרי קוד שיעזרו בשינוי מ- COCO ל- YOLO ומצאתי קוד שעושה כמעט מה שהייתי צריך באתר ששמוMedium .

על מנת להבין איך הקוד עובד צריך קודם להבין את הפורמטים של YOLO ו- COCO.

YOLO דורש שכל אחד מה- annotation שלו יהיה בפורמט הבא:

<object-class> <x\_center> <y\_center> <width> <height>

כאשר:

object-class: מספר מ- 0-6 (במקרה שלי) שמראה לאיזה תווית ה-annotation שייך

x\_center: מספר עשרוני (float) שמתאר את המרכז של המלבן התוחם בציר ה- x (מנורמל מ- 0-1)

y\_center: מספר עשרוני (float) שמתאר את המרכז של המלבן התוחם בציר ה- y (מנורמל מ- 0-1)

width: רוחב המלבן התוחם (מספר עשרוני) מנורמל מ- 0-1

height: גובה המלבן התוחם (מספר עשרוני) מנורמל מ- 0-1

לעומת זאת, COCO נותן את ה- annotation שלו בפורמט הבא:

category, image\_id, x\_start, y\_start, width, height

כאשר:

category: קטגורית האובייקט (במקרה שלי התוויות הן בקטגוריות-על)

image\_id: מספר סידורי של התמונה שבה נמצא ה- annotation

x\_start: מיקום תחילת האובייקט בציר ה- x (לא מנורמל)

y\_start: מיקום תחילת האובייקט בציר ה- y (לא מנורמל)

width: רוחב התמונה (לא מנורמל)

height: גובה התמונה (לא מנורמל)

בנוסף, הגרסה הרשמית של YOLO, שבשלב זה לא הייתי בטוח אם אשתמש בה או בגרסה של YOLO ב- TensorFlow, דורשת שלכל תמונה יהיה קובץ משל עצמה שמכיל את ה- annotations שנמצאים בה.

לכן הייתי צריך ליצור קוד שעושה את הדברים הבאים:

* עובר על כל אחת מהתמונות ושולף כל אחת מה- annotation שלה
* על כל annotation, מוצא את קטגורית-העל שמתאימה לו
* על כל annotation, שולף את ערכי התיבות התוחמות ומשנה אותם לפורמט YOLO
* שומר על כל תמונה קובץ טקסט שמכיל את כל ה- annotation שלה בשורות נפרדות

למטרה זו יצרתי שלוש פונקציות:

הפונקציה הראשונה נקראה

all\_annot\_for\_image

הפונקציה קיבלה כפרמטר מספר סידורי של תמונה והחזירה את כל ה- annotations שלה בפורמט מתאים. הפונקציה בדקה איזה annotations מתאימים לתמונה, העבירה עליהם את המניפולציה הדרושה ולבסוף החזירה את כל ה- annotation של התמונה כמחרוזת.

על מנת לבצע מניפולציה זו יצרתי פונקציה ששמה convert\_annot שנועדה לקחת annotation אחד ולהפוך אותו לפורמט YOLO.

הפונקציה עושה מספר דברים:

* לוקחת את ה- annotation ומעבירה אותו למצב ביניים שבו התיבה התוחמת שלו נראית כך (x\_start,y\_start,x\_end,y\_end)
* מעבירה את ה- annotation החדש דרך פונקציה שלקחתי מאתר Medium שמשנה את התיבות התוחמות שלהן לפורמט YOLO
* לוקחת את הקטגוריה של ה- annotation ומגלה את קטגורית-העל שלה
* בודקת את קטגורית-העל שלה כנגד dictonary שיצרתי כדי להפוך אותה למספר
* לבסוף מחברת את הכל ל- string לפי פורמט YOLO

לפונקציה מאתר Medium קוראים convert\_label והיא עובדת כך:

* יוצרת משתנים ל- x\_min,x\_max,y\_min ו- y\_max לפי המידע שהיא מקבלת מהפרמטרים ומוודאה שהם בסדר הנכון
* יוצרת משתנים בשביל הנורמליזציה שהם 1 / גודל התמונה
* מחשבת את מרכז הריבוע בכך שהיא מחברת את המיקומים ההתחלתיים ואת המיקומים הסופיים ומחלקת ב- 2
* מגלה אורך ורוחב של הריבוע בכך שהיא מחסירה את המיקום הסופי מהמיקום ההתחלתי של הריבוע
* מבצעת נורמליזציה (מכפילה את המשתנים שגילתה ב- 1 / גודל התמונה)
* מחזירה את ארבעת הערכים של התיבה התוחמת בפורמט YOLO

בסוף הקוד אני עובר על כל התמונות, מריץ על כל אחת את הפונקציה all\_annot\_for\_image ושומר את מה שהיא מחזירה בקובץ image\_id.txt.

הערה: הקוד של שלוש הפונקציות שתיארתי השתנה במהלך הפרויקט, אתאר כיצד בחלק האימון.

לאחר מכן הייתי צריך לנסות אימון, אך כאשר הרצתי אימון נוצרו בעיות. את הבעיות עצמן אתאר בפרק האימון, אך אפרט את הסיבות להן כאן, כאשר חשוב להבהיר שמדובר בהשערה ואיני בטוח אם הן נכונות.

הסיבה הראשונה שלדעתי יכלה ליצור בעיות באימון היא העובדה שיתכן שחלק מהאורך והרוחב של התיבות התוחמות קטן מדי. YOLOv3 יכול לזהות עד גודל 13\*13, אך כאשר הסתכלתי על התמונות שלי ראיתי bounding boxes קטנות יותר. עם זאת, לדעתי זה לא הגורם לבעיה.

הסיבה השניה שיכלה ליצור בעיות באימון היא האפשרות שיש טעות באחד מחלקי הקוד שמשנים את מסד הנתונים, ככל הנראה במהלך או לאחר שינוי גודל התמונות. אפשרות אחת היא בקוד שינוי הפורמט של COCO ל- YOLO. עם זאת, גם לאחר שבחנתי את הקוד מספר פעמים לא מצאתי שגיאה.

מכיוון שלאחר מחקר רב וחקירה של מספר דרכים ליישם את YOLO לא מצאתי פתרון, החלטתי שאין מנוס מהחלפת מסד הנתונים במסד נתונים חדש, שאותו בניתי בעצמי, כפי שיפורט להלן.

## **מסד נתונים שני: מסד נתונים נבנה אישית**

### הסבר על תוכנות תיוג

### כאמור, YOLO דורש גם תמונות וגם annotations. במקרה של מסד נתונים צריך לסמן את התמונות באופן ידני ובכדי לעשות זאת בצורה יעילה אפשר להשתמש בתוכנות שמאפשרות לסמן ריבועים בשימוש בממשק משתמש פשוט.

ב- GitHub אפשר למצוא מספר תוכנות מסוג זה כגון:

[label studio](https://github.com/heartexlabs/label-studio#included-templates-for-labeling-data-in-label-studio)- תוכנה פתוחה המאפשרת לעשות תיוג לתמונות, אודיו, טקסט ועוד.

[labelImg](https://github.com/tzutalin/labelImg)- תוכנה פתוחה המשתמשת ב- GUI פשוט לצורך תיוג של תמונות בלבד.

בחרתי להשתמש ב- labelImg מהסיבות הבאות:

* ה- GUI פשוט וקל להבנה, דבר שמאפשר עבודה מהירה
* יש אפשרות לשימוש ב- hotkeys שמשפרים מהירות עוד יותר
* וחשוב מכך - יש אפשרות להעביר את הסימנים אוטומטית לפורמט YOLO, דבר שעוזר להוריד את כמות ה- preprocessing ולמנוע טעויות

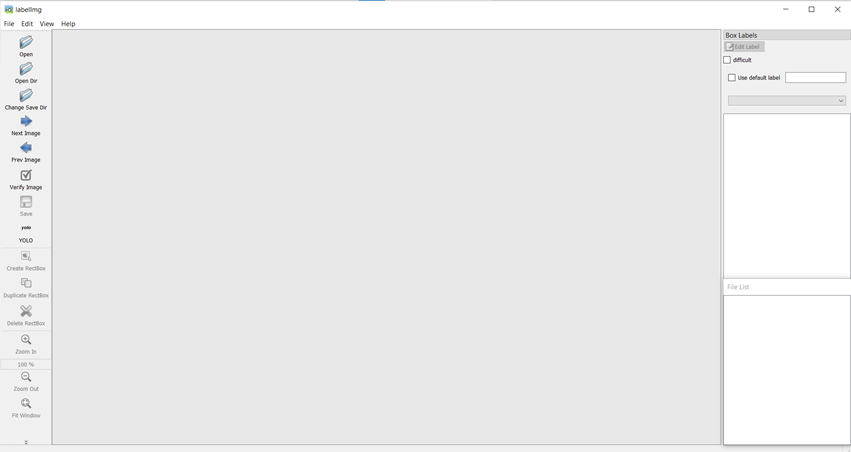
את labelImg מריצים באמצעות שימוש בפקודה הבאה:

labelImg [IMAGE\_PATH] [PRE-DEFINED CLASS FILE]

Image\_path: המיקום של התמונות

pre-defined-class-file: המקום של קובץ טקסט המתאר את כל התוויות

ה- GUI של labelImg נראה כך (תמונה מ- GitHub):



התוכנה עובדת כך:

* לוחצים על open dir ופותחים את התיקייה עם התמונות
* לוחצים על open save dir ובוחרים את התיקייה שבה רוצים לשמור את קבצי הטקסט של ה- annotations
* בכל תמונה לוחצים על create rect או על a בשביל להתחיל annotation
* מסמנים ריבוע ובוחרים את ה- class שלו מרשימת פופ-אפ שמופיעה
* שומרים
* חוזרים על שלבים 3-5 לכל אחת מהתיבות בתמונה
* ממשיכים לתמונה הבאה עם d או next image
* בסיום סוגרים את התוכנה

### בניית מסד הנתונים

לאחר מציאת תוכנה מתאימה לתיוג אפשר להתחיל לעבוד על בניית מסד נתונים. החלטתי להוריד את קטגוריית ה- uknown מאחר והגעתי למסקנה שאם המכונה לא יודעת לאיזה פח צריך לזרוק משהו היא לא תסמן אותו. כפי שאפרט בהמשך, איסוף המידע בכל קטגוריה היה שונה.

#### זכוכית

את תמונות הזכוכית אספתי באמצעות חיפוש ב- Google. השתמשתי בתוכנה שמאפשרת להוריד את כל התמונות באתר אינטרנט בשם Batch Image Downloader. החיפוש שהשתמשתי בו הוא glass bottle והצלחתי לאסוף 100 תמונות, שבחלקן היו מספר פריטים, דבר שככל הנראה שיפר את יכולת המכונה לזהות פריטי זכוכית שונים באותה תמונה.

למרות שתכננתי לאסוף 100 תמונות לכל תווית, מצאתי שכמות זו אינה מספיקה לזיהוי ולכן החלטתי להוסיף תמונות ובשלב זה הוספתי תמונות של 15 צלחות ו- 15 כוסות.

תמונת דוגמה ל- annotation:



#### מתכת

את 100 תמונות המתכת מצאתי במסד הנתונים בו השתמשתי בעבודת הקיץ בלמידת מכונה. היו בו 500 תמונות ועל מנת לבחור 100 תמונות רנדומליות כתבתי קוד שמשתמש בספריית glob בכדי לשלוף את כל התמונות וב- Numpy לצורך בחירת 100 מתוכן באופן רנדומלי.

לעובדה שבחרתי תמונות ממסד הנתונים של עבודת הקיץ יתרונות וחסרונות. היתרון הוא שהכרתי את התמונות וידעתי איך לתייג אותן והחיסרון הוא שהתמונות נבנו בשביל Classification רגיל, כלומר יש פריט אחד בכל תמונה והוא מכסה חלק גדול ממנה.

לאחר שבחנתי כיצד המודל עובד עם התמונות החדשות, מצאתי בעיות בזיהוי מספר פריטים (כפי שחזיתי) ולכן הוספתי עוד 25 תמונות, בעיקר של פחיות שתיה, שהן פסולת המתכת המשמעותית ביותר. בכל תמונה היו מספר פחיות והדבר בהחלט שיפר את יכולת הזיהוי.

תמונת דוגמה ל- annotation:

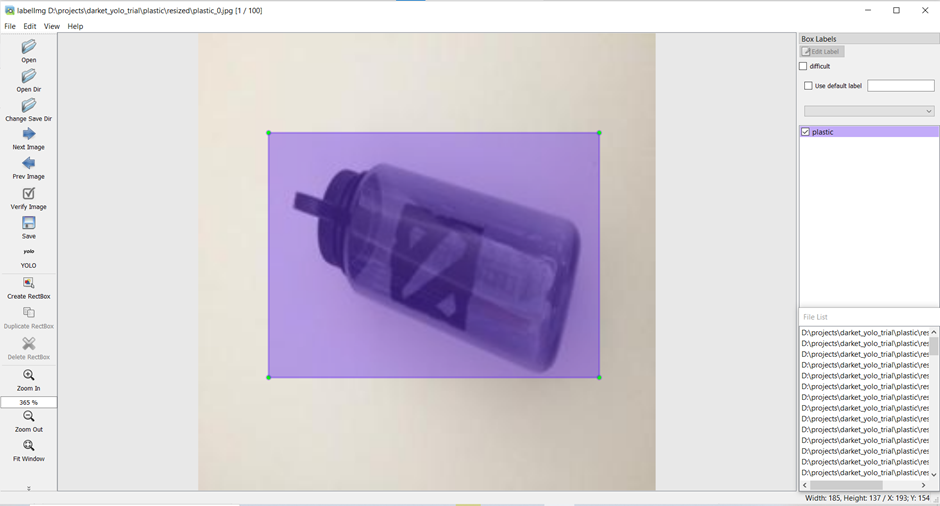


#### פלסטיק

גם את תמונות הפלסטיק אספתי ממסד הנתונים של עבודת הקיץ. רוב התמונות היו של בקבוקים ופקקים מפלסטיק, אך מאחר והתמונות הכילו פריט אחד, חזיתי גם הפעם בעיות בזיהוי מספר פריטים.

קטגוריה זו היתה קשה יותר מכיוון שהיא כוללת פריטים מסוגים רבים והמודל צריך היה לזהות בקבוקים, קופסאות, פקקים ויתכן אף חפצים כגון לגו או ניילון. סיבה נוספת לקושי בקטגוריה זו היתה האפשרות שהמודל יתבלבל בין פלסטיק למתכת ולכן החלטתי לבצע מספר בדיקות על קופסאות מפלסטיק. המודל זיהה נכונה את המיקום, אך חשב שהן מתכת ולכן הוספתי 25 תמונות נוספות של קופסאות פלסטיק בצבעים וזוויות שונות.

תמונת דוגמה ל- annotation:

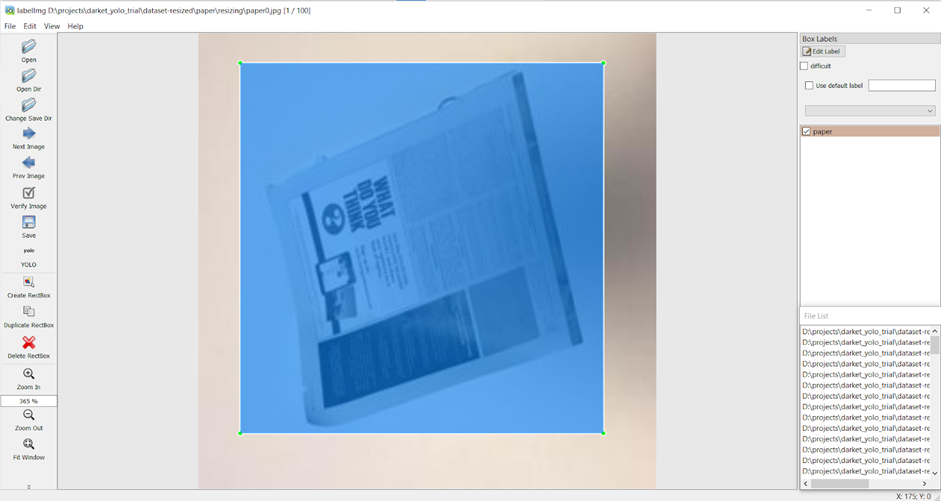


#### נייר

גם בקטגוריה זו אספתי 100 תמונות, בעיקר של עיתונים וכן מגזינים וספרים. התמונות נלקחו מחיפוש ב- Google וכללו כיתוב בעברית ובאנגלית.

במהלך הבדיקה התברר כי בשימוש בצילומי מסך מטלפון נייד המודל יוצר bounding box הכוללת את כל התמונה עם זיהוי של נייר. מאחר והפרויקט לא עסק בצילומי מסך מטלפון, לא נדרשתי לתקן בעיה זו. כמו כן, מאחר וקטגוריה זו אינה מגוונת וקלה יחסית לזיהוי, לא הרגשתי צורך להוסיף תמונות נוספות.

תמונת דוגמה ל- annotation:



#### קרטון

בקטגוריה זו אספתי תמונות של קופסאות ואוריגמי מקרטון באמצעות חיפוש ב- Google, על מנת לבדוק האם הדבר יעיל יותר משימוש במסד הנתונים של עבודת הקיץ, אך מצאתי 93 תמונות בלבד בגודל מתאים. פעולת הסימון היתה מעט קשה כי כאשר בתמונה היו מספר פריטים מחוברים, הייתי צריך להחליט מה נחשב פריט אחד ומה נחשב כמספר פריטים.

גם בקטגוריה זו היתה אפשרות שהמודל יתבלבל בין קרטון לפלסטיק, אך מאחר וקיים שוני בצבעים, צפיתי שלא תיווצר בעיה. היתרון הוא שיש לי מספיק מקום לשימוש ב- RGB ולא הייתי צריך להשתמש ב- Grayscale, כפי שעשיתי בעבודת הקיץ.

תמונת דוגמה ל- annotation:

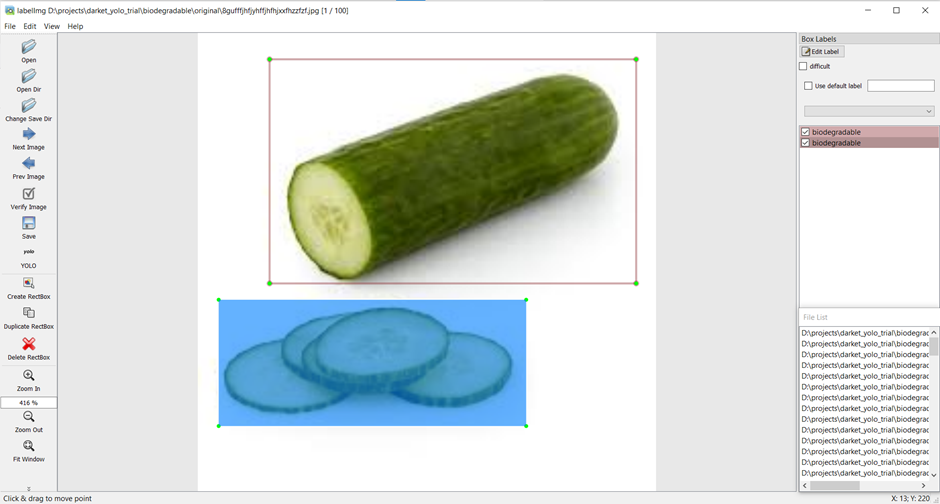


#### מתכלה

בקטגוריה זו אספתי 100 תמונות באמצעות חיפוש ב- Google, מתוכן 70 תמונות של פירות וירקות ו- 30 תמונות של אגוזים, עוגות ופשטידות.

מאחר ומדובר בקטגוריה שונה, לא צפיתי קונפליקטים מול יתר הקטגוריות ואכן בבדיקה ראשונית המודל היה בטוח יותר לגבי קטגוריה זו. למרות שלא בחנתי את יכולת המודל לזהות שילוב של מספר פריטים, לדעתי לא אמורה להיווצר בעיה.

תמונה דוגמא ל- annotation :



# הסבר על **YOLO**

בדרך הנפוצה של זיהוי אובייקטים (רשת Classification שמסתיימת בשכבת Softmax) אפשר לזהות רק אובייקט אחד שהוא ככל הנראה האובייקט היחיד בתמונה ואין דרך לדעת את מיקומו של האובייקט. בפרויקט זה רציתי לזהות מספר אובייקטים שיכולים להיות בעלי תוויות שונות ולמטרה זו השתמשתי בשיטת למידת מכונה ששמה YOLO.

YOLO (ראשי תיבות של You Only Look Once) הוצעה ב- 2015 במאמר [You Only Look Once: Unified, Real-Time Object0 Detection - arXiv](https://arxiv.org/abs/1506.02640). הרעיון הבסיסי של YOLO הוא פשוט - מחלקים כל תמונה ל- grid בגודל מסויים ובודקים האם בתוך כל תא יש פריט שמתאים לאחת מהתגיות שנבחרו. למטרה זו לרוב או לכל התמונות במסד הנתונים צריכה להיות bounding box לכל אחד מהפריטים שמתאימים לאחת מהתגיות. לכן הייתי צריך לבצע סימון של התמונות או להשיג מסד נתונים מסומן.

שימוש ב- bounding boxes יוצר שתי בעיות פוטנציאליות. הבעיה הראשונה היא כאשר פריט נמצא בתוך שני תאים או יותר של ה- grid. במקרה זה YOLO משתמש בשני אלגוריתמים בכדי לפתור את הבעיה. הראשון הוא IOU (ראשי תיבות של Intersection over Union) - אלגוריתם זה לוקח שתי bounding boxes ומחזיר את היחס בין שטח החיתוך לבין סכום השטחים שלהן.

האלגוריתם השני - Non Max Suppression מתחיל בכך שהוא מתעלם מכל ה- bounding boxes שלא עברו סף הסתברות מסוים, לאחר מכן לוקח את ה- bounding box עם ההסתברות הגבוהה ביותר בתא ונפטר מכל ה- bounding boxes שה- IOU שלהם איתו עובר סף מסוים. אלגוריתם זה עובד אך ורק במהלך ה- Inference ולא במהלך ה- Training .

הבעיה השניה היא כאשר קיים יותר מפריט אחד בתא. במקרה זה YOLO משתמש ב- Anchors. כל תא מתחלק למספר Anchors וכל Anchor יכול לזהות פריט נפרד. לדוגמה, תמונה שבה בתא אחד יש אדם עומד וכלב, Anchor אחד יכול להיות ארוך וצר (כדי לזהות את האדם) ו- Anchor שני יכול להיות רחב וקצר (כדי לזהות את הכלב). YOLO מספק default Anchors שאפשר להשתמש בהם באימון ולכן במקרה שתיווצר בעיה אוכל לחשב את ה- Anchors מחדש.

פונקצית המחיר של YOLOv4 בנויה משלושה חלקים:

* Binary Cross Entropy על מנת לדעת האם יש אובייקט בכל אחד מה-Anchors, השוואה בין הניחוש לאמת
* Binary Cross Entropy על מנת לבדוק איזה אובייקט יש בכל אחד מה- Anchors
* IOU על מנת לחשב את המרחק של ה- bounding box שהמודל ניחש מה - bounding box האמיתי. ה- IOU החליף את Mean Squared Error בגרסאות הקודמות

כאמור, YOLO מחלק את התמונה פעמיים - פעם ראשונה ל- grid שגודל כל אחד מתאיו הוא 32. במקרה של תמונה בגודל 256\*256 יש grid בגודל 8\*8 ופעם שניה ל- Anchors ולכן כאשר הרשת מחזירה את הניחוש שלה (y predict) היא מחזירה אותו במבנה הבא (n,8,8,6,g). כש- n הוא מספר התמונות ו- g הוא 6 או 5+ מספר התוויות בתלות השימוש ב- one-hot.

בגלל גודל כל תא ב- grid, המודל מקבל אך ורק תמונות שהגודל שלהן (גובה ורוחב) מתחלק ב-32. לכן היה צורך לשנות את גודל התמונות. הגרסה המקורית משתמשת ב- 416\*416, אך במקרה שלי העדפתי להשתמש ב- 256\*256 בשביל שהאימון יהיה מהיר יותר.

הגרסה של YOLO שבה אני משתמש היא YOLOv4-tiny בגלל שהאימון שלו הרבה יותר מהיר (לפעמים אפילו פי 30 יותר מהיר). המודל משתמש ב- 6 Anchors במקום 9 ומבצע 2 זיהויים בגדלים שונים במקום 3. החיסרון הוא דיוק מופחת - ⅔ ממה ש- YOLOv4 מבצע ברוב המקרים, אך העדפתי לאמן את המודל במהירות גדולה יותר ולא לחכות מספר ימים לכל אימון של המודל.

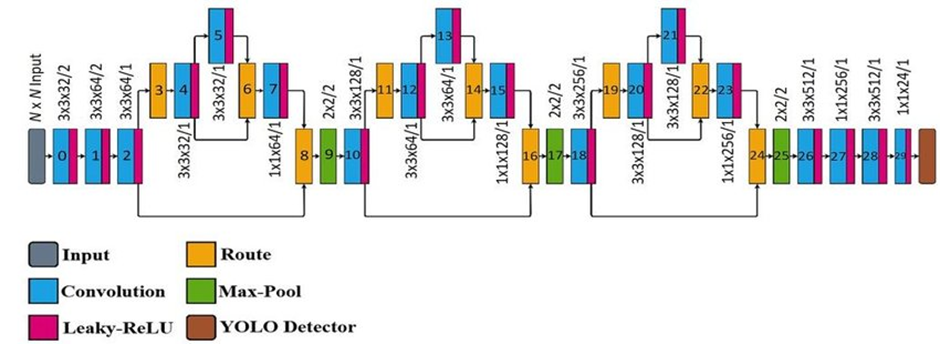
הדרך שבה YOLO מחשב את דיוק המודל הוא באמצעות מדד ששמו MAP.

MAP (ראשי תיבות של Mean Average precision) הוא ממוצע של דיוק ה- bounding boxes בכל התמונות. לכל תמונה יש AP שזה ממוצע הדיוק בתמונה עצמה. ה- AP מורכב משני מדדים ששמם Precision ו - Recall.

Precision הוא היחס בין כמות ה- bounding boxes שנמצאו במדויק לכמות ה- bounding boxes האמיתיים. לעומת זאת, Recall הוא היחס בין כמות ה- bounding boxes שנמצאו (כולל אלו שאינם מדוייקים) לכמות ה- bounding boxes האמיתיים.

לרוב ה- Precision וה- Recall מחושבים באמצעות שימוש ב- thresholds שונים, ה- AP לכל תמונה הוא ממוצע של היחס בין Precision ו- Recall וה- MAP הוא ממוצע של ה- AP של כל התמונות.

תיאור גרפי של המודל:



בחלקים של המודל שהם Route, המודל בוחר מחצית מה- Input ושולח אותה לשכבות הבאות.

השכבות של ה- YOLO Detector הם שכבת Convolution של 3\*(מספר הקטגוריות+5) פילטרים. הכפולה 3 מייצגת את כמות ה- Anchors בזיהוי ומספר הקטגוריות+5 מייצג את כמות התוויות ומיקום ה- bounding box. חשוב לציין שיש 2 זיהויים במודל ובציור מתואר רק אחד.

בנוסף בכל פעם שיש יותר מחץ אחד שמגיע לשכבה מסויימת מתבצע לפני זה concatantion שהוא חיבור כל תוצאות השכבות שנכנסות לשכבה.

# אימון וחקירת **HyperParameters**

## הקדמה על האימון

על מנת ליצור מודל YOLO שיוכל לזהות פסולת צריך לאמן אותו באמצעות הדוגמאות. למטרה זו היו לי מספר אפשרויות. האפשרות הראשונה היתה לעבוד עם ספריה שיוצרת מודל YOLO באמצעות TensorFlow. לאחר חיפוש ב- GitHub מצאתי מספר מודלים מוגדרים ב- TensorFlow אך לכל אחד מהם היו בעיות.

הראשון שלקחתי יצר בעיות עם ה- loss function והיה מוגדר עם TensorFlow 1. השני שבחרתי היה מוגדר ב- TensorFlow 2, אך יצר בעיות עם ה- loss function שלא הצלחתי לפתור. הבעיות היו קשורות למספר פונקציות שלא היו מוגדרות ועדיין הפעילו אותן. המודל השלישי שמצאתי יצר בעיות עם הגדרת נתוני האימון (x\_train ו- y\_train) והיה צורך לשנות את הדרך שבה ה- annotations נשלחים למודל, מהדרך שתיארתי בפרק מסד הנתונים, לשני קבצים שבהם בכל שורה יש מיקום תמונה וכל ה- annotations של אותה תמונה מופרדים ברווחים.

לאחר מכן עברתי לשימוש בספריה ששמה Darknet שהם היוצרים המקוריים של YOLO. על מנת לגרום למודל לעבוד הייתי צריך לבצע עוד preprocessing.

Darknet דורש שלכל תמונה יהיו ה- annotations שלה בקובץ נפרד ויהיו שני קבצים בשביל ה -test וה- train שיכילו את ה- path לכל אחת מהתמונות שבהן הם ישתמשו. יצרתי את הקבצים בשימוש בקוד שלוקח כל אחת מהתמונות ובודק אותן מול ליסט של אינדקסים.

שינוי נוסף שהיה צריך לעשות הוא בקובץ הקונפיגורציה של YOLO. קובץ הקונפיגורציה קובע את המבנה של הרשת של YOLO. לדוגמה, הוא קובע אילו שכבות יהיו שכבות Convolution ואילו שכבות יהיו שכבות זיהוי. השינוי הראשון שהיה צריך לעשות הוא לשנות את גודל התמונות של ה- Input ל- 256\*256 ולאחר מכן לשנות את שכבות הזיהוי של המודל, בהן צריך לזהות את כמות ה- classes מה- default שהוא 80 ל- 6. לאחר מכן היה צורך לשנות את שכבות ה- Convolution לפני הזיהוי, בהן הפילטרים היו צריכים להיות classes+5)\*3).

## הסבר על האימון

המטרה העיקרית באימונים היא לבחון מספר קונפיגרציות שונות של הרצת YOLO ולשנות את ה- Batch size, אורך הלמידה (באיטרציות) ובעיקר את ה-learning rate. בכל אימון אפרט את הגדרות האימון, גרף האימון והמסקנות ממנו.

גרף אימון של YOLO מכיל שני חלקים, החלק הראשון הוא גרף ה- loss של המודל, שמחושב כל מאה איטרציות והחלק השני הוא הדיוק, שמחושב פעם ראשונה אחרי 1000 איטרציות ולאחר מכן כל 100 איטרציות נוספות. הסבר על חישוב הדיוק פורט בפרק המודל.

## אימונים

### אימון ראשון

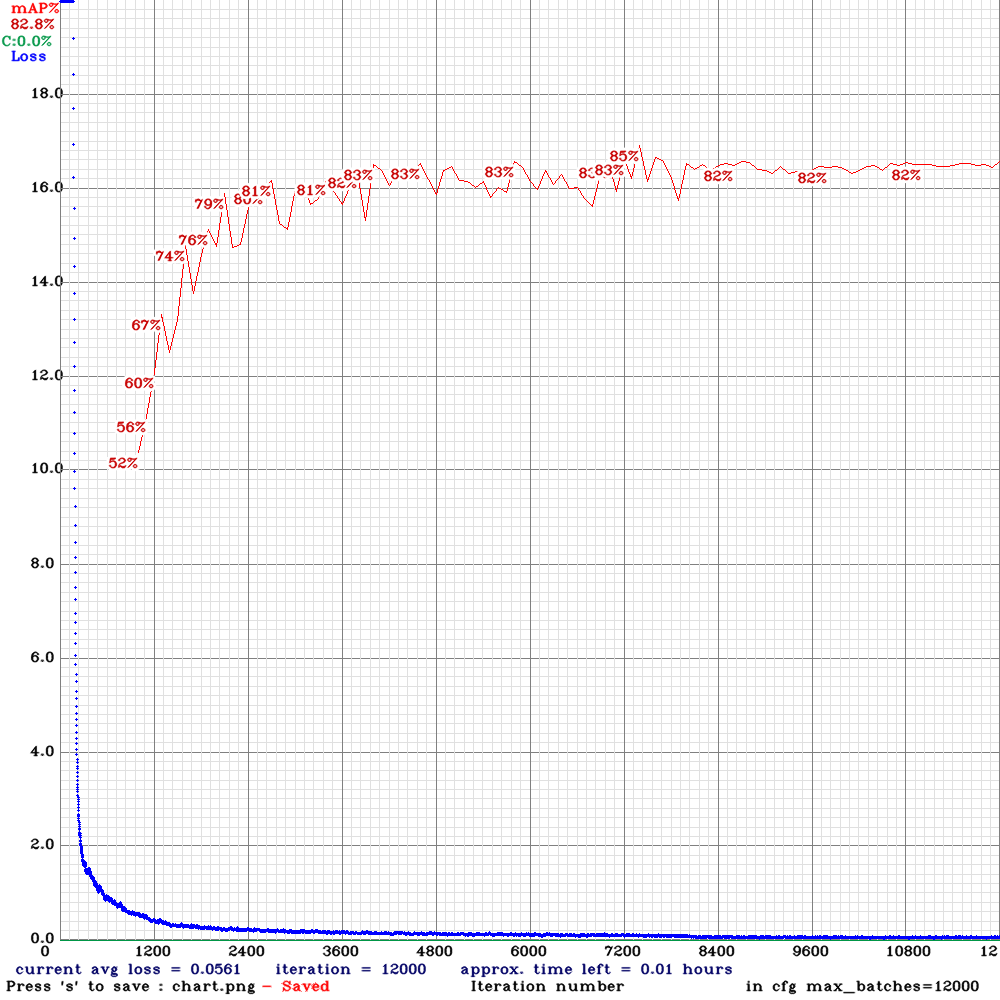
האימון הראשון הוא אימון בשימוש בהגדרות המומלצות של Darknet. אימון זה הוא הקו הבוחן של כל שאר האימונים, כלומר אליו אשווה את האימונים הנוספים.

ההגדרות של האימון:

* learning\_rate=0.00261
* max\_batches = 12000
* batch=64

max\_batches זה כמות האיטרציות, batch זה ה- Batch size ו- learning\_rate זה ה- learning rate.

הגרף של האימון:



האימון על פי Google Colab נמשך שעה ו- 11 דקות. ל- 85% אחוזי זיהוי זהו זמן סביר ביותר, בעיקר בהתחשב בכך שאימון ב- YOLOv4 היה נמשך בין 9 (הערכה שלי) ל- 20 שעות (לפי YOLO).

לפי הגרף, שיפוע ירידת המחיר בהתחלה היה גבוה מאוד, כי המחיר היה גבוה מאוד בהתחלה (מעל 18). אחרי 200 איטרציות הירידה הפכה להיות מתונה יותר, עד שלא ניתן היה לשפר את המחיר. לעומת זאת הדיוק (MAP) התחיל ב- 52 וזינק ל- 79 ואז החלה מגמה של עליה וירידה עד שהמודל הגיע לדיוק מקסימלי של 85%.

לדעתי, לפי מגמת העליה והירידה של ה- MAP, יתכן שהמודל עובר overfitting, אך אחוזי הזיהוי עדיין טובים מאד ויתכן שישנה אפשרות להוריד את כמות האיטרציות באימון (אבדוק זאת בהמשך).

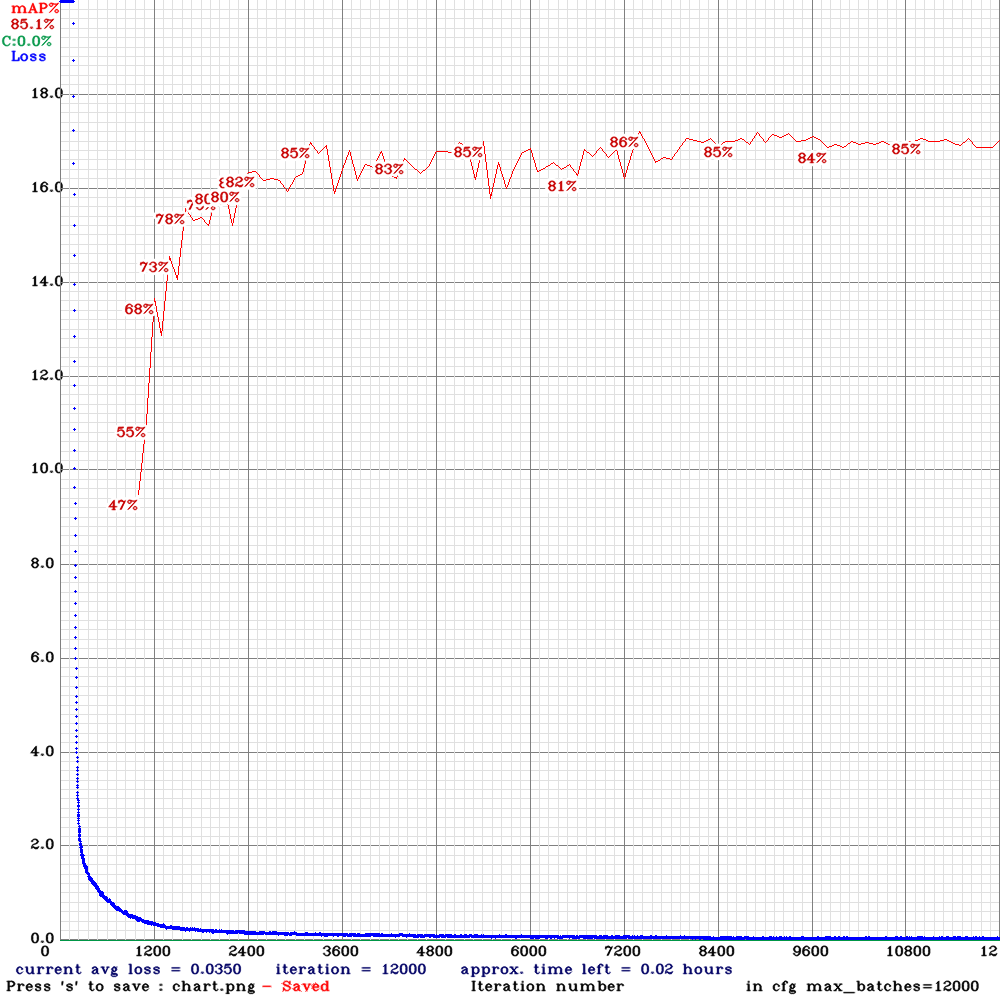
### אימון שני

שני האימונים הבאים יבדקו את השפעת ה- Batch size על האימון של YOLO. ההשערה שלי היא שככל שה- Batch size יהיה גדול יותר יקח ל- YOLO יותר זמן לאמן את המודל, מכיוון שהוא מקבל יותר תמונות כל איטרציה. עם זאת, יתכן שהמודל יהיה יותר מדוייק.

ההגדרות של האימון:

* learning\_rate=0.00261
* max\_batches = 12000
* batch=128

הגרף של האימון:



כפי שניתן לראות, המחיר המינימלי היה נמוך יותר מהאימון הראשון. אפשר גם לראות שה- MAP הגיע לדיוק מקסימלי של 86%, כלומר מבחינת דיוק, המודל הנוכחי דומה מאד למודל הקודם, אך מעט טוב יותר.

לעומת זאת, הזמן שנדרש לאימון במודל זה היה כמעט כפול. הדבר לא מפתיע מכיוון שכל איטרציה המודל היה צריך לעבור על פי 2 תמונות.

יש לציין, כי מודל זה הגיע ל- MAP של 85% תוך 3,000 איטרציות לעומת 7,500 במודל המקורי ויתכן שאפשר להפוך את אימון המודל למהיר יותר דרך הורדת כמות האיטרציות. אבחן זאת בהמשך.

לסיכום, הקונפיגורציה של אימון זה פחות יעילה, אך היא פותחת אפשרויות לבדיקות נוספות, כגון לאמן פחות איטרציות ואף לנסות לשנות את ה- learning rate. אפשרויות אלו אבחן באימונים עתידיים.

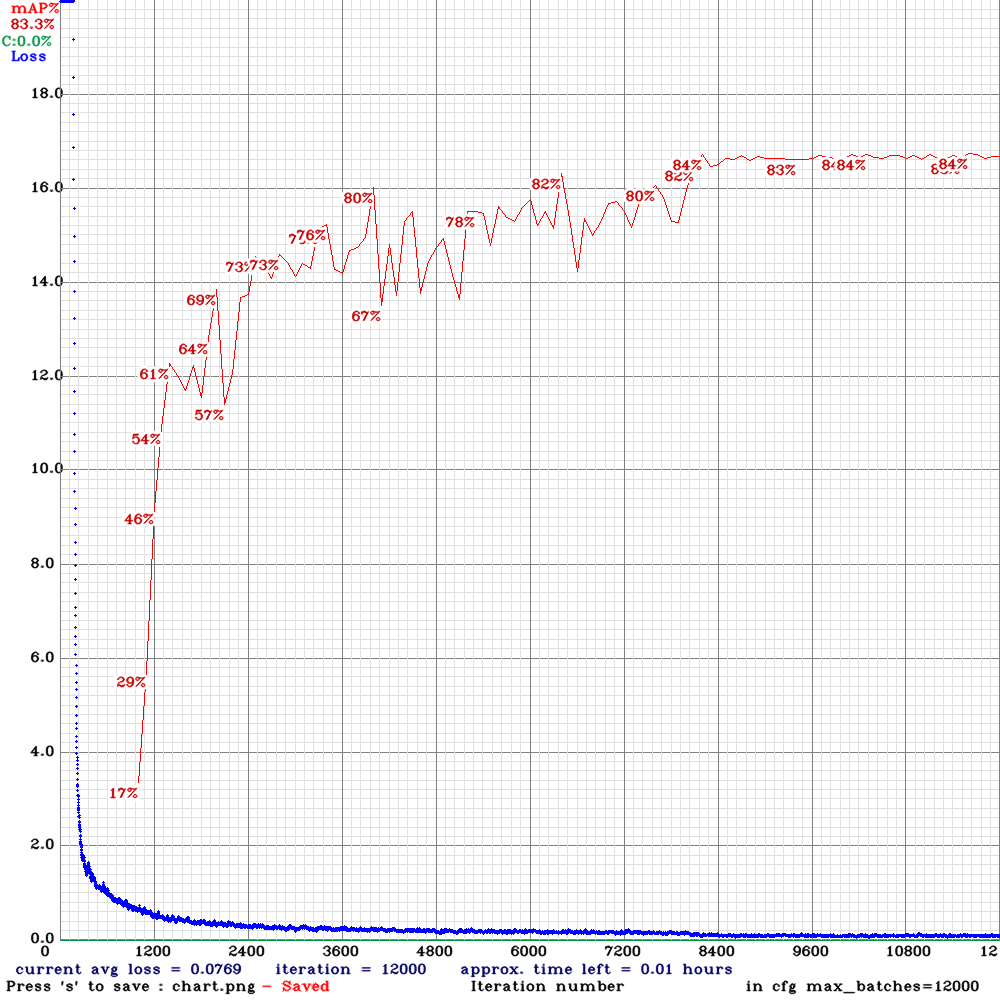
### אימון שלישי

האימון השלישי היה הפוך מהאימון השני, שכן שהקטנתי את ה- Batch size במקום להגדילה. צפיתי שגם המודל יגיב ההיפך, כלומר יתן תוצאות פחות טובות, אך האימון יהיה יותר מהיר.

ההגדרות של האימון:

* learning\_rate=0.00261
* max\_batches = 12000
* batch=32

הגרף של האימון:



לפי הגרף ניתן לראות שאימון זה היה פחות מוצלח מהקודמים מבחינת loss ו- MAP. האימון היה יותר מהיר מהאימון השני (79 לעומת 119 דקות), אך איטי יותר מהאימון הראשון (79 לעומת 71 דקות).

אימון זה אינו טוב במיוחד, הוא גם פחות מדויק משני האימונים הקודמים וגם פחות מהיר מהאימון הראשון ולכן אם הייתי צריך לבחור מבין שלושת האימונים שעשיתי עד כה, לא הייתי בוחר בו.

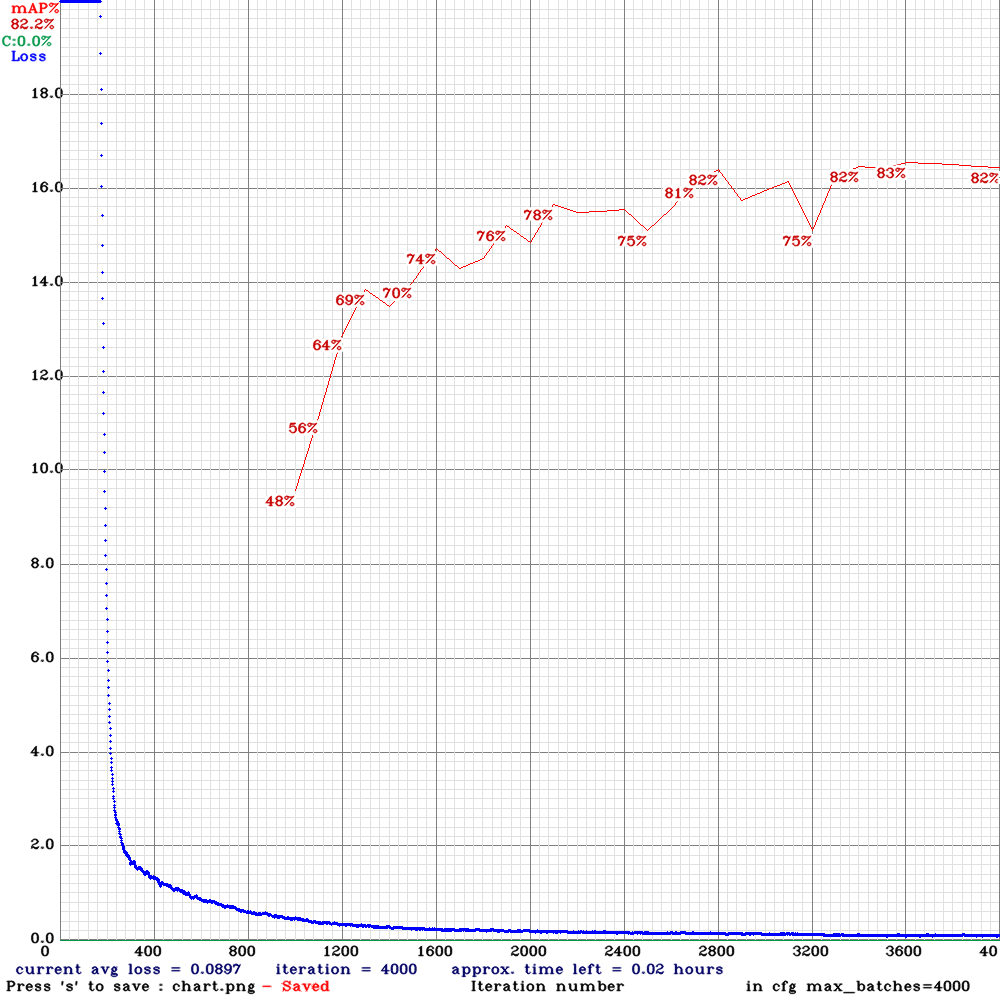
### אימון רביעי

האימון הרביעי הוא האימון שהצעתי באימון השני, כלומר Batch size של 128 ו- 4000 איטרציות. לפי דעתי זה יהיה האימון היעיל ביותר, גם מבחינת מהירות, מאחר שהאימון רץ רק כשליש מהאיטרציות וגם מבחינת דיוק על פי גרף האימון השני.

ההגדרות של האימון:

* learning\_rate=0.00261
* max\_batches = 4000
* batch=128

הגרף של האימון:



המודל לא היה טוב כפי שציפיתי (82% במקום 85%) והסיבה לכך לדעתי היא בפרמטר שעוד לא הזכרתי ששמו steps, שהוא פרמטר המתאר ל- YOLO איך לשנות את ה- learning rate במהלך האימון. הפרמטר מורכב מ- 2 מספרים - הראשון הוא 80% מכמות האיטרציות והשני 90% מכמות האיטרציות. המודל מהאימון השני שהגיע ל- 85% MAP ב- 4000 איטרציות ביצע זאת עם steps שהתאימו ל- 12,000 איטרציות והמודל שגרף זה מתאר מאומן עם steps שמתאימים לאימון של 4000 איטרציות.

המודל לא רע ומצליח להגיע ל- 83% MAP ב- 47 דקות! זהו המודל המהיר ביותר עד כה, אך יתכן ששינוי ה- steps יעזור למודל להגיע לתוצאות של המודל השני. אבחן זאת באימון הבא.

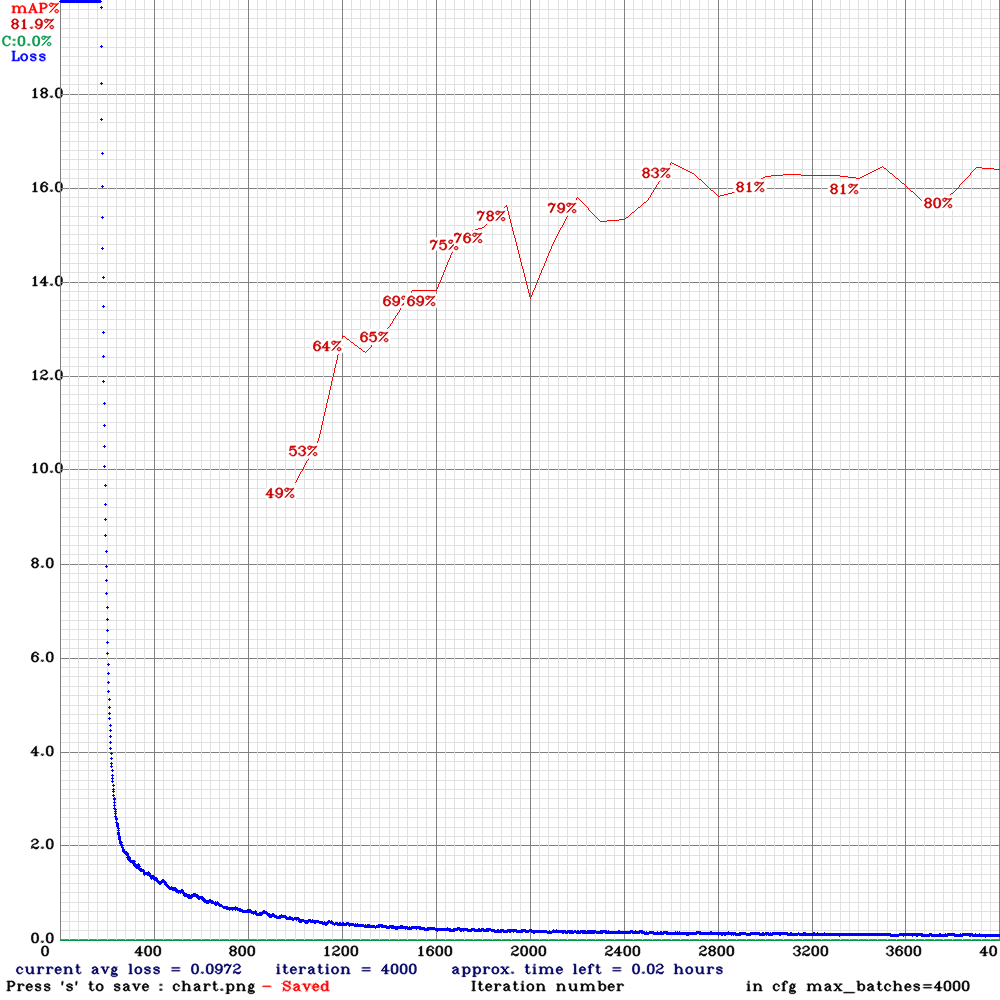
### אימון חמישי

מטרת אימון זה היא לבחון האם שינוי ה- steps ישיג את אותן התוצאות כמו במודל השני ב- 4000 איטרציות.

ההגדרות של האימון:

* learning\_rate=0.00261
* max\_batches = 4000
* batch=128
* steps:9600 ,10800

הגרף של האימון:



באופן מפתיע, ביצוע המודל לא השתפר והמקסימום שהוא הגיע אליו הוא 83% MAP. לעומת זאת, המודל היה מהיר (47 דקות), אך יתכן שה- steps הוא לא ההבדל היחיד מהמודל השני. בסך הכל, המודל עובד יחסית טוב.

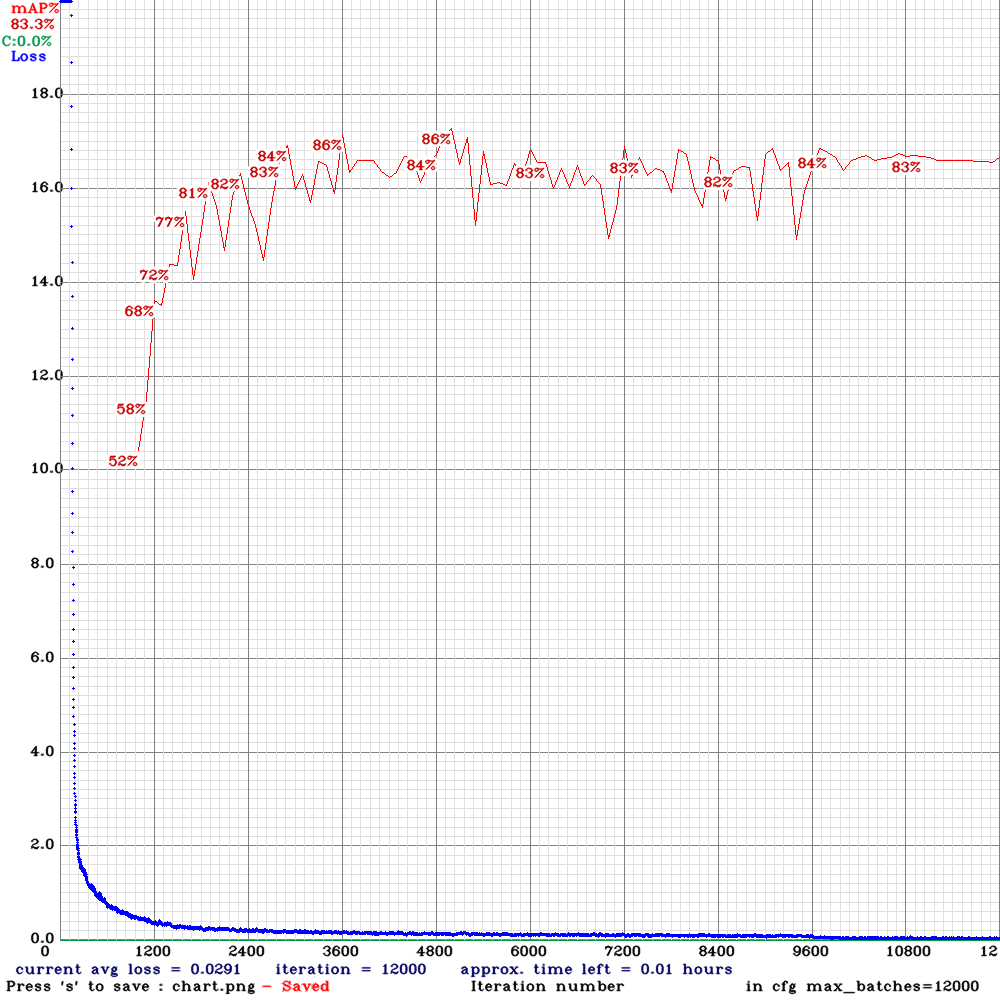
### אימון שישי

מטרת האימונים הבאים היא לבחון את השפעת ה- learning rate. הניסויים הבאים יהיו ב- Batch size 64. על מנת לקבוע את ה- learning rates של שלושת האימונים הבאים אצור מספר רנדומלי (float) ואכפיל אותו כל פעם ב 10 בחזקת מינוס 3 עד מינוס 1.

ההגדרות של האימון:

* learning\_rate=0.0064
* max\_batches = 12000
* batch=64

הגרף של האימון:



מהגרף הנוכחי ניתן לראות ששיפוע הירידה של ה- loss גבוה יותר מהאימון הראשון, למרות שהוא מגיע לרמת דיוק דומה של 86% MAP, אך הרבה יותר מהר.

ה- learning rate הנוכחי אינו גבוה בהרבה מה- learning rate הסטנדרטי של 0.0026 והוא עוזר למודל ללמוד מהר יותר. יתכן שהגדלת ה- learning rate עוד תפגע בתוצאות אלו או תוביל ל- convergence מהיר יותר.

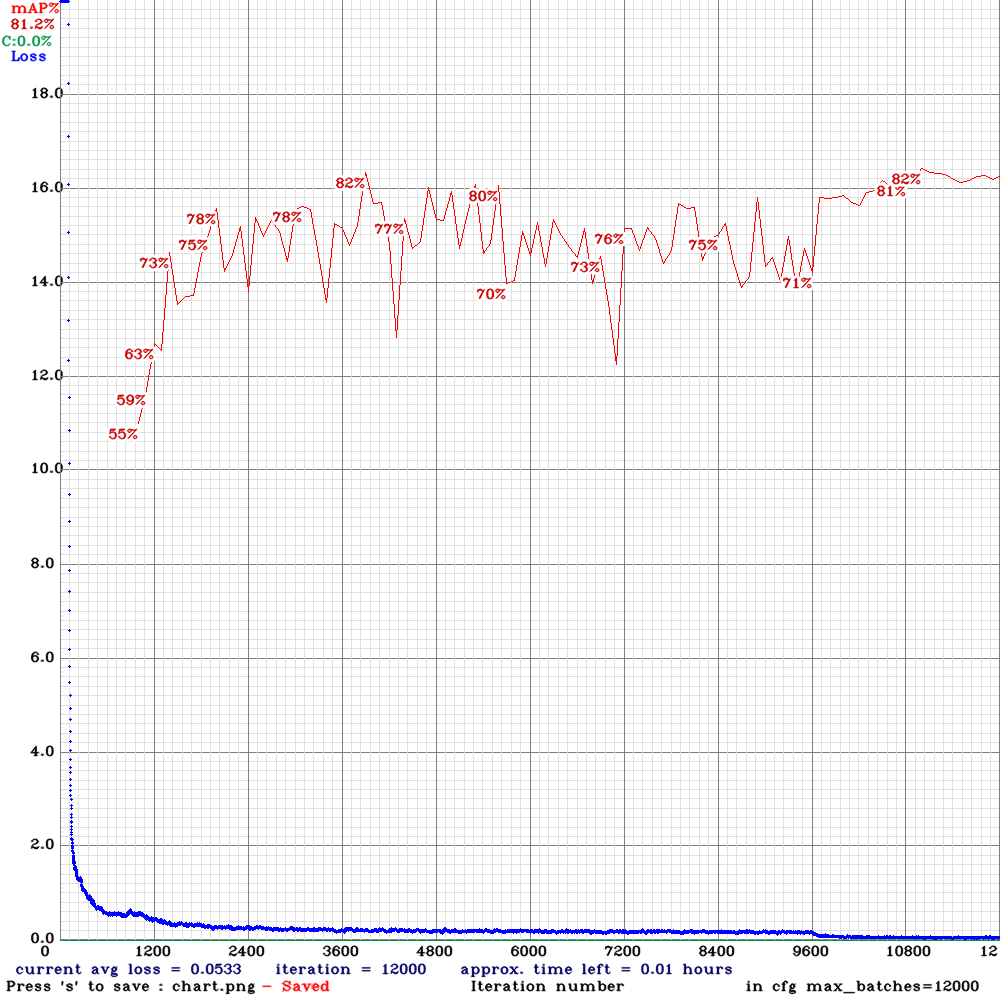
### אימון שביעי

האימון הזה הוא האימון השני בבחינת ה- learning rate. אימון זה יהיה עם learning rate של 0.036.

ההגדרות של האימון:

* learning\_rate=0.036
* max\_batches = 12000
* batch=64

הגרף של האימון:



כפי שניתן לראות learning rate גבוה פוגע ככל הנראה באימון, לפחות כך זה נראה מהמודל הנוכחי. יתכן שהסיבה לכך היא שבכל איטרציה המשקלים משתנים יותר מדי והמודל לא מסוגל ללמוד כמו שצריך

על מנת לקבוע בצורה חד משמעית את השפעת learning rate גבוה, יש צורך בניסוי נוסף של learning rate גבוה יותר, כפי שאעשה באימון הבא.

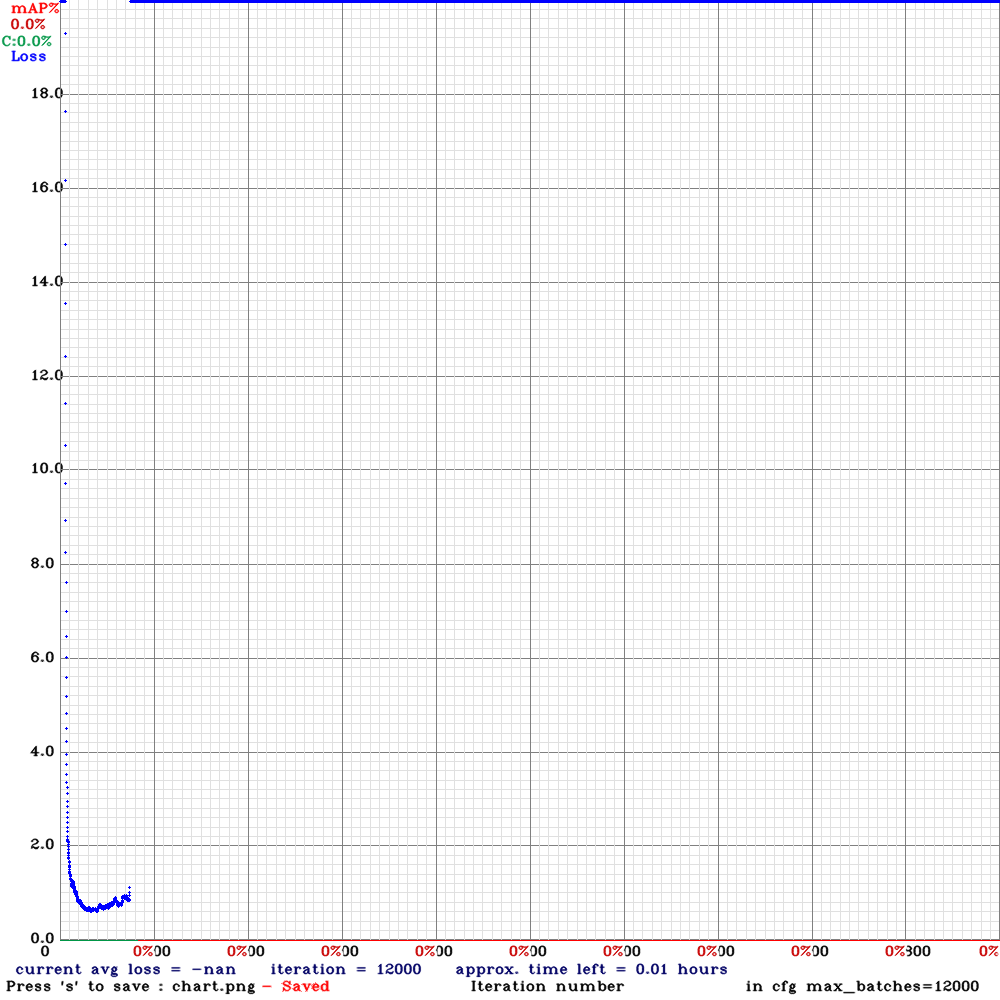
### אימון שמיני

האימון השמיני יהיה האימון עם learning rate של 0.62 שהוא הגבוה ביותר שאנסה. אני חוזה שהמודל יאמן את עצמו בצורה לא טובה, בדומה למודל הקודם ויתכן שאף יהיה לא יציב.

ההגדרות של האימון:

* learning\_rate=0.62
* max\_batches = 12000
* batch=64

הגרף של האימון:



ניתן לראות שהאימון היה מאד לא יציב והסתיים ב -NAN (ראשי תיבות של not a number). מעניין לראות שהמודל התחיל את האימון היטב ואז באופן פתאומי קפץ ל-NAN. ההשערה שלי היא שהיה באחד החישובים log 0 או חילוק באפס ואז כאשר חושב ה- loss function הכל הפך ל- NAN.

מסקנתי משלושת אימונים אלו היא שה- learning rate הסטנדרטי של YOLO הוא יחסית טוב ולכן חובה לשמור אותו בטווח של חזקה של עשר כמו במודל הסטנדרטי.

### אימון תשיעי

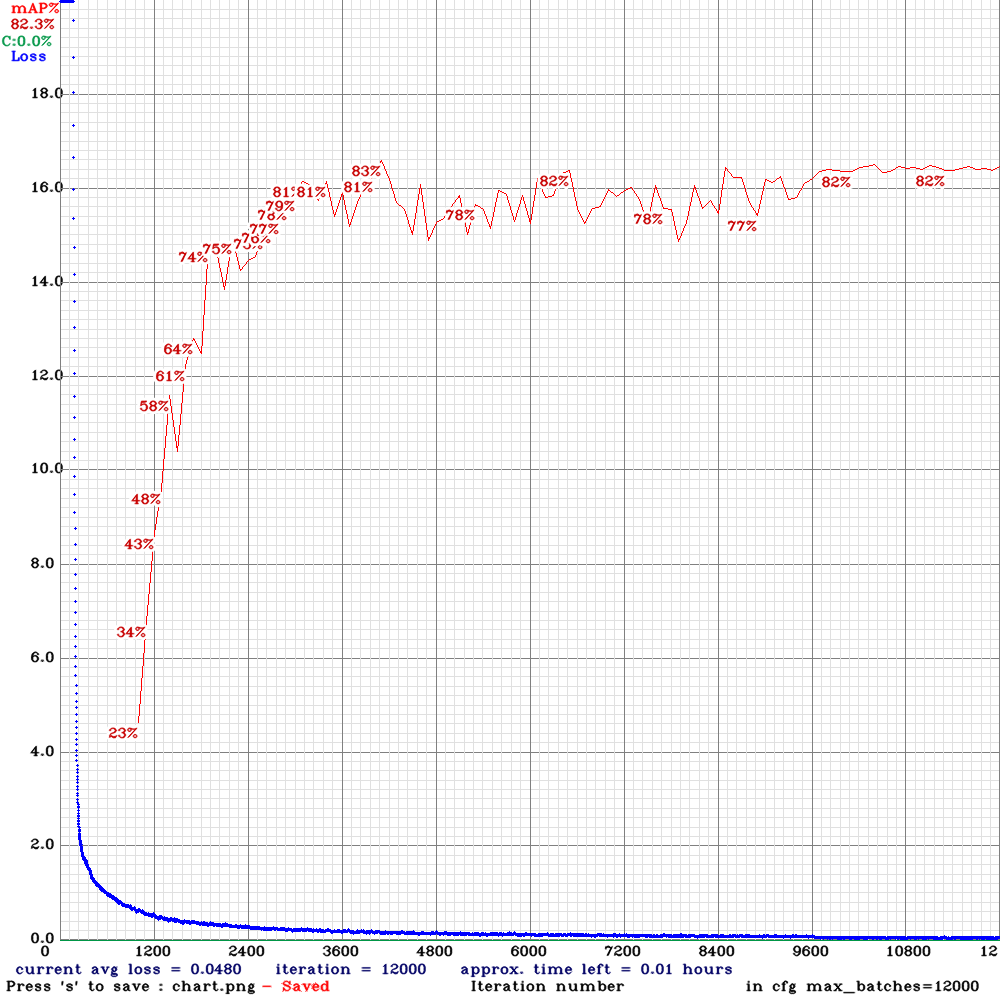
האימון הבא קשור ל- Anchors, שהם פרמטר שקובע איך YOLO מגלה מספר פריטים שנמצאים באותו תא ב- grid (הסבר מפורט נמצא בפרק המודל).

ל- YOLO קיימים Anchors סטנדרטיים שבהם השתמשנו עד כה, אך ישנה אפשרות לחשב את ה- Anchors הטובים ביותר למידע באמצעות Darknet.

ההגדרות של האימון:

* learning\_rate=0.00261
* max\_batches = 12000
* batch=64
* אנקורים: 48,110, 126,100, 76,196, 145,223, 209,157, 231,230

הגרף של האימון:



כפי שניתן לראות שינוי ה- Anchors לא שיפרו את ביצועי המודל אלא פגעו בו. ה- MAP המקסימלי באימון הוא 82% לעומת 85% באימון המקורי ולדעתי הסיבה לכך היא שה- Anchors המקוריים של YOLO מכסים את התמונות בצורה מדוייקת יותר.

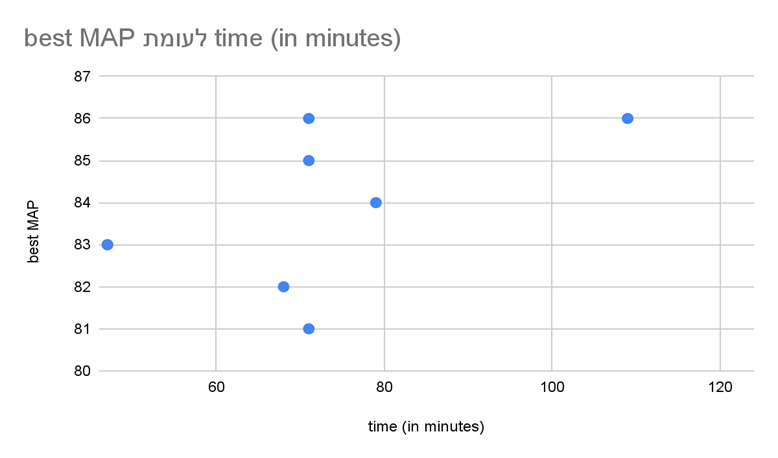
## סיכום האימונים

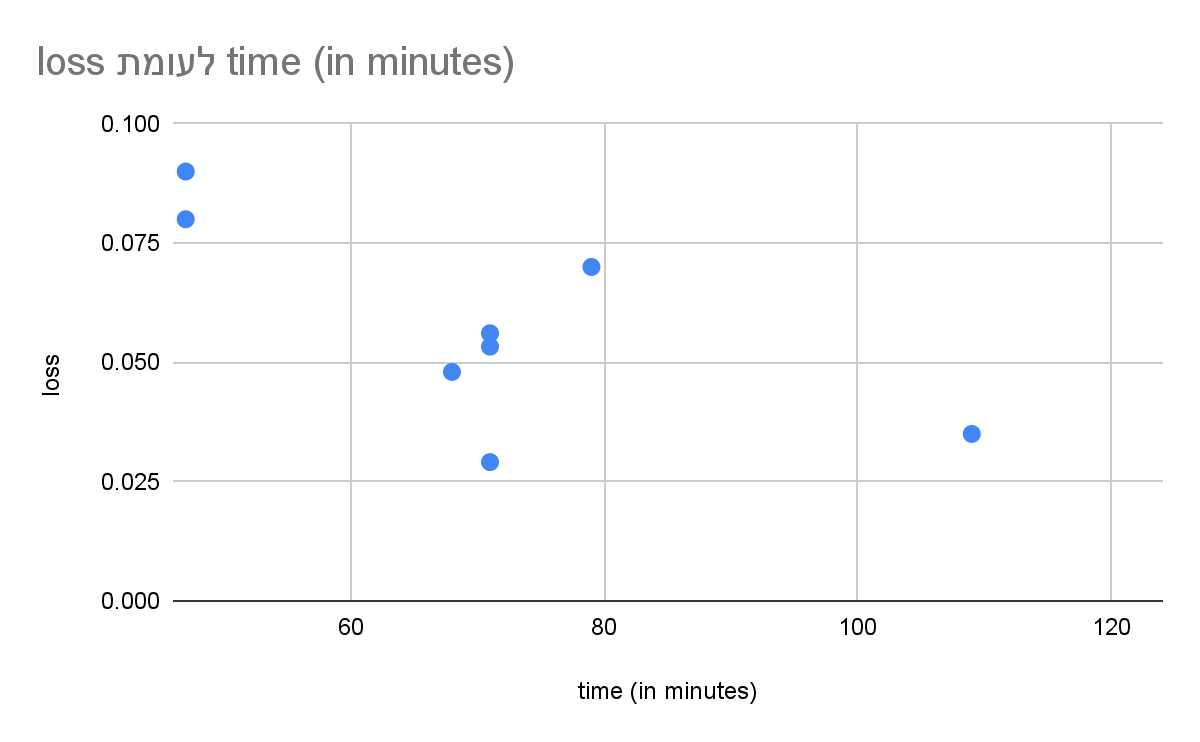
## באופן מפתיע, שינוי ה- HyperParameters לא השפיע הרבה על הדיוק של המכונה (חוץ במקרה של ה-NAN). לעומת זאת, זמן האימון השתנה מאד בשינוי הפרמטרים ולכן במקרה של אימון מכונה מההתחלה יש לקחת את השילוב הטוב ביותר של זמן ודיוק (MAP).

את סיכום האימונים ניתן לראות בטבלה הבאה:

|  |  |  |  |
| --- | --- | --- | --- |
| trial number | time (in minutes) | loss | best MAP |
| 1 | 71 | 0.0561 | 85 |
| 2 | 109 | 0.035 | 86 |
| 3 | 79 | 0.07 | 84 |
| 4 | 47 | 0.08 | 83 |
| 5 | 47 | 0.09 | 83 |
| 6 | 71 | 0.0291 | 86 |
| 7 | 71 | 0.0533 | 81 |
| 8 | 123 | -NAN | NAN |
| 9 | 68 | 0.048 | 82 |

וכן בשני הגרפים הבאים:



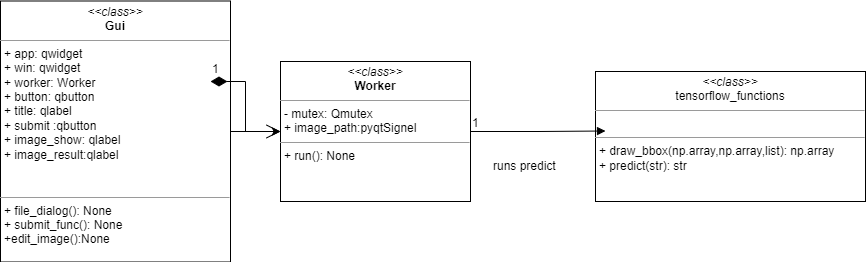


# תיאור יצירת היישום

התוצר הסופי של עבודת הגמר הוא אפליקציה בעלת ממשק משתמש (GUI), המאפשר למשתמש לקחת קובץ תמונה שבו יש פריטי פסולת ולקבל תמונה חדשה שבה כל פריט מסומן בריבוע תוחם שמעליו יש את פח המיחזור שאליו הוא צריך להיזרק.

על מנת ליצור את ממשק המשתמש השתמשתי בספרייה ששמה pyqt5 שהיא גרסה של Python לתוכנה ששמה QT שנועדה ליצירת ממשקי משתמש שיכולים לעבוד בכל מערכת הפעלה. היתרון של התוכנה היא הקלות שבה יוצרים את ממשק המשתמש עצמו (התווית, הכפתורים וכו') והעובדה שאפשר לבצע שם multithreading יחסית בקלות.

תרשים זרימה:



ה- GUI מפעיל את ה- worker class שמטרתו ליצור thread בשביל שהאפליקציה לא תתקע, ולאחר מכן ה- worker מריץ את פונקצית ה- predict ב- TensorFlow\_functions.

על מנת להשתמש במודל הייתי צריך להפוך את מודל ה- Darknet שלי למודל TensorFlow בגלל שהתקנת Darknet על Windows קצת מסובכת. למטרה זו השתמשתי בספריה שמצאתי ב- GitHub בשם

TensorFlow-YOLOv4-tflite שעושה זאת. לאחר מכן הייתי צריך לכתוב את הקוד שלוקח את המודל שנוצר ומשתמש בו בשביל ליצור את ה- bounding boxes ולאחר מכן מצייר אותן על התמונה.

למטרה זו כתבתי שתי פונקציות: draw\_bbox בשביל הציור ו- predict שמפעילה את המודל וקוראת ל- draw\_bbox בשביל לצייר את ה- bounding boxes.

predict עובדת כך:

* מקבלת מיקום של תמונה (file path), קולטת את האורך והגובה של התמונה, מקטינה אותה ל- 256\*256 ומבצעת נורמליזציה
* מעבירה את התמונה דרך המודל ומקבלת את ה- y\_predict
* מריצה את אלגוריתם Non Max Suppression
* קוראת ל- draw\_bbox בשביל שתצייר את ה- bounding boxes על התמונה
* מגדילה את התמונה חזרה לגודל המקורי שלה, שומרת ומחזירה את המיקום החדש

draw\_bbox עובדת כך:

* מקבלת תמונה, bounding boxes, רשימה של תוויות ומשתנה בוליאני שקובע האם להראות את התוויות בתמונה
* יוצרת רשימה של צבעים שאיתם היא עובדת
* מבטלת את הנורמליזציה של ה- bounding boxes באמצעות הכפלה בגודל התמונה
* עוברת על ה-bounding boxes ומציירת כל אחת בצבע
* אם צריך לצייר את התווית בתמונה, הפונקציה כותבת את הטקסט מעל ה- bounding boxes
* מחזירה את התמונה

מדריך למפתח

*#GUI.py*

*#functions and classes for the GUI*

import os

from PyQt5.QtCore import QThreadPool, QRunnable, pyqtSlot, QThread, QObject, pyqtSignal, QMutex, QUrl, Qt

from PyQt5.QtMultimedia import QMediaPlayer, QMediaContent

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import \*

from PIL import Image

import sys

from pathlib import Path

from tensorflow\_functions import predict

import time

*# threading to stop the application from freezing when process browse\_button is clicked*

class Worker(QThread):

   image\_path = pyqtSignal(str)  *# object that signal when the thread is done, and returns the image\_path for the gui*

def \_\_init\_\_(self, parent=None):

       self.\_mutex = QMutex()  *# QMutex is used to prevent mutation of the data used by the thread*

super().\_\_init\_\_()  *# calling Qthread \_\_init\_\_()*

*# function that runs the Thread and calls the predict function*

def run(self):

       self.\_mutex.lock()  *# locking the data*

img = Image.open(file\_save\_path)  *## opning the image*

if not Path("saves").exists():

           os.mkdir("saves")  *# creating save folder if it doesn't exist*

img\_path\_for\_pred = Path("saves/" + time.strftime("%Y%m%d-%H%M%S") + ".jpg")  *# crating path for save*

img.save(str(img\_path\_for\_pred))  *# saving*

img\_path\_for\_show = predict(

           str(img\_path\_for\_pred))  *# calling prediction function see tensorflow\_function code part*

self.image\_path.emit(img\_path\_for\_show)  *# telling the gui the code is done*

self.finished.emit()  *# QThread requires this line*

self.\_mutex.unlock()  *# unlocking the data used by the thread*

*# Class that manages the gui*

class Gui:

   def \_\_init\_\_(self):

       global file\_save\_path  *# file path of chosen image, used later*

file\_save\_path = None

*# most of the following lines builds the Parts of the gui*

*# QApplication is for the application, Qlabel is for showing text and images, QPushButton is for buttons*

*# setFont is function to set the font,*

*# #setText sets the text of move places the widgit in the gui and*

*# connect connects a function to somthing that happens (usually click*

self.app = QApplication(sys.argv)  *# creating application*

self.win = QMainWindow()

       self.win.setGeometry(200, 200, 1200, 1200)

       width, height = self.win.frameGeometry().width(), self.win.frameGeometry().height()

       self.win.setWindowTitle("School project Uri Bracha-Gui app")

       self.worker = Worker()

       self.title = QLabel(self.win)  *# title of the project*

self.title.setFont(QFont("Times", 14, QFont.Bold))

       self.title.setText("School project Uri Bracha-Gui app")

       self.title.adjustSize()

       self.title.move((width // 4) + 50, 0)

       self.chosen\_image\_name = QLabel(self.win)  *# chosen\_image\_name for showing chosen image name*

self.chosen\_image\_name.setFont(QFont("Times", 14))

       self.chosen\_image\_name.setText("choose an image to process:")

       self.chosen\_image\_name.move(50, 75)

       self.chosen\_image\_name.adjustSize()

       self.browse\_button = QPushButton(self.win)  *# browse\_button to allow browsing*

self.browse\_button.setText("Browse..")

       self.browse\_button.setFont(QFont("Times", 14))

       self.browse\_button.adjustSize()

       self.browse\_button.move(350, 75)

       self.browse\_button.clicked.connect(self.file\_dialog)

       self.file\_text = QLabel(" file not chosen ...", self.win)  *# label for showing name of chosen file*

self.file\_text.setFont(QFont("Times", 14))

       self.file\_text.adjustSize()

       self.file\_text.move(500, 75)

       self.process\_button = QPushButton(self.win)  *# button to activate processing function(model)*

self.process\_button.setText("process")

       self.process\_button.setFont(QFont("Times", 14))

       self.process\_button.adjustSize()

       self.process\_button.clicked.connect(self.process\_func)

       self.process\_button.move(350, 175)

       self.image\_show = QLabel(self.win)  *# label that  shows the chosen image*

self.image\_show.move(350, 175)

       self.image\_result = QLabel(self.win)  *# label that shows image after model processing*

self.image\_result.setFont(QFont("Times", 14))

       self.win.show()  *# activating gui*

sys.exit(self.app.exec\_())  *# code for exiting gui when close is clicked*

*# function that runs when browse\_button is clicked, opens file dialog for choosing image file*

*# takes the file and shows the image on the gui*

def file\_dialog(self):

       global file\_save\_path  *# this variable keeps the file path to the chosen image,*

*# it is global becuase it is used in Worker class*

file, check = QFileDialog.getOpenFileName(None, "QFileDialog.getOpenFileName()", "")  *# opens file dialog*

path = Path(file)

       file\_save\_path = file  *# sets the global variable*

self.file\_text.setText(path.name)  *# sets the label*

pix = QPixmap(file)  *# create map of pixels for use in label*

*# height and width regularization for pixel map*

if pix.height() > 350:

           pix = pix.scaled(pix.width(), 350, Qt.KeepAspectRatio, Qt.FastTransformation)

       if pix.width() > 350:

           pix = pix.scaled(350, pix.height(), Qt.KeepAspectRatio, Qt.FastTransformation)

       self.process\_button.move(350, 200 + pix.height())  *# moves the button so it isn't covered*

self.image\_show.setPixmap(pix)

       self.image\_show.adjustSize()

*# function that runs when the process button is clicked, it's only is use is activating the thread*

def process\_func(self):

       self.image\_result.move(350, 250 + 350)

       self.image\_result.clear() *#clear the image show of the imgae*

self.image\_result.setText("working ...")

       self.image\_result.move(350,250+350)

       self.image\_show.adjustSize()

       self.worker.image\_path.connect(self.edit\_image)  *# sets the function that is run after the processing is done*

self.worker.start()  *# starts the running of the thread*

*# function that shows the image with the bounding boxes in gui, file\_image\_path is the path to the new image*

def edit\_image(self, file\_image\_path):

       pix = QPixmap(file\_image\_path)  *# creates pixel map from path*

*# height and width regularization for pixel map*

if pix.height() > 350:

           pix = pix.scaled(pix.width(), 350, Qt.KeepAspectRatio, Qt.FastTransformation)

       if pix.width() > 350:

           pix = pix.scaled(350, pix.height(), Qt.KeepAspectRatio, Qt.FastTransformation)

       self.image\_result.clear() *#clear label of text*

self.image\_result.setText("") *#making sure text is cleared*

self.image\_result.setPixmap(pix)  *# sets label to map*

self.image\_result.adjustSize()

       self.image\_result.move(350, self.browse\_button.height() + self.browse\_button.y() + pix.height() + 200)*#moving it so it doesn't cover other things*

if \_\_name\_\_ == '\_\_main\_\_':

   g = Gui()

*TensorFlow\_functions.py*

*#functions that manage interaction with images and TensorFlow model for use with GUI*

import colorsys

import random

import TensorFlow as tf

import cv2

import numpy as np

from pathlib import Path

from PIL import Image, ImageDraw, ImageColor

from TensorFlow.python.saved\_model import tag\_constants

import sys

import os

*#function that takes an image and an array of bounding boxes sized (n,5) and returns an image with bounding boxes drawn*

def draw\_bbox(image, bboxes, classes=[**"glass"**, **"metal"**, **"plastic"**, **"paper"**, **"cardboard"**, **"biodegradable"**],

show\_label=True):

num\_classes = len(classes)

image\_h, image\_w, \_ = image.shape

*#define colors for bounding boxes and the labels*

hsv\_tuples = [(1.0 \* x / num\_classes, 1., 1.) for x in range(num\_classes)]

colors = list(map(lambda x: colorsys.hsv\_to\_rgb(\*x), hsv\_tuples))

colors = list(map(lambda x: (int(x[0] \* 255), int(x[1] \* 255), int(x[2] \* 255)), colors))

random.seed(0)*#random seed for the same colors each time*

random.shuffle(colors)

random.seed(None)

out\_boxes, out\_scores, out\_classes, num\_boxes = bboxes

*#for loop for drawing boxes*

for i in range(num\_boxes[0]):

if int(out\_classes[0][i]) < 0 or int(out\_classes[0][i]) > num\_classes: continue

coor = out\_boxes[0][i]

coor[0] = int(coor[0] \* image\_h)

coor[2] = int(coor[2] \* image\_h)

coor[1] = int(coor[1] \* image\_w)

coor[3] = int(coor[3] \* image\_w)

fontScale = 0.5

score = out\_scores[0][i]

class\_ind = int(out\_classes[0][i])

bbox\_color = colors[class\_ind]

bbox\_thick = int(0.6 \* (image\_h + image\_w) / 600)

c1, c2 = (int(coor[1]), int(coor[0])), (int(coor[3]), int(coor[2]))

cv2.rectangle(image, c1, c2, bbox\_color, bbox\_thick)

*#label drawing*

if show\_label:

bbox\_mess = **'%s: %.2f'** % (classes[class\_ind], score)

t\_size = cv2.getTextSize(bbox\_mess, 0, fontScale, thickness=bbox\_thick // 2)[0]

c3 = (c1[0] + t\_size[0], c1[1] - t\_size[1] - 3)

cv2.rectangle(image, c1, (int(np.float32(c3[0])), int(np.float32(c3[1]))), bbox\_color, -1) *# filled*

cv2.putText(image, bbox\_mess, (c1[0], int(np.float32(c1[1] - 2))), cv2.FONT\_HERSHEY\_SIMPLEX,

fontScale, (0, 0, 0), bbox\_thick // 2, lineType=cv2.LINE\_AA)

return image

*# function that takes a file path to a single image, predicts bounding boxes inside of it saves and returns new file\_path*

def predict(image\_path):

model = tf.saved\_model.load(**"model/YOLOv4-tiny-model"**, tags=[tag\_constants.SERVING]) *#loading model*

orig\_img = cv2.imread(image\_path) *#loading\_image*

orig\_img = cv2.cvtColor(orig\_img, cv2.COLOR\_BGR2RGB)

orig\_width, orig\_height,\_ = orig\_img.shape

orig\_img = cv2.resize(orig\_img, (256, 256))*#resizing*

img = np.array(orig\_img).astype(np.float32) *#copy of image so that the original image stay intact*

infer = model.signatures[**'serving\_default'**]

*#data\_prepration*

img = img / 255

img = np.expand\_dims(img, 0)

img = tf.constant(img)

*#predication*

pred\_bbox = infer(img)

for key, value in pred\_bbox.items():

boxes = value[:, :, 0:4]

pred\_conf = value[:, :, 4:]

boxes, scores, classes, valid\_detections = tf.image.combined\_non\_max\_suppression(

boxes=tf.reshape(boxes, (tf.shape(boxes)[0], -1, 1, 4)),

scores=tf.reshape(

pred\_conf, (tf.shape(pred\_conf)[0], -1, tf.shape(pred\_conf)[-1])),

max\_output\_size\_per\_class=50,

max\_total\_size=50,

iou\_threshold=0.2,

score\_threshold=0.5

)

pred\_bbox = [boxes.numpy(), scores.numpy(), classes.numpy(), valid\_detections.numpy()]

*#drawing bounding boxes and saving image*

new\_img = draw\_bbox(orig\_img, pred\_bbox, show\_label=True) *#new image*

new\_img = Image.fromarray(new\_img)*#changing image to pillow format*

new\_img=new\_img.resize((orig\_height,orig\_width))

save\_name = **"saves/"** + str(Path(image\_path).stem) + **"\_output.jpg"** *#file name*

new\_img.save(save\_name) *#saving*

return save\_name

# מדריך למשתמש

הוראות התקנה

לצורך התקנת ה- GUI צריך לעשות את הדברים הבאים:

* להוריד את תוכנת git
* להריץ את הפקודה הבאה:

git clone https://github.com/uribracha2611/YOLOv4\_tiny\_waste\_detection.git

* לפתוח קונסולה (cmd or powershell or bash) ולהריץ את הפקודה הבאה:

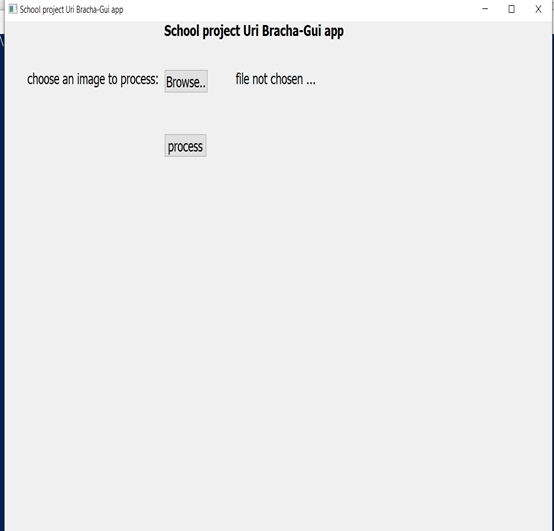
pip install -r requirements.txt

* להריץ בתוך אותה קנסולה python gui.py

הערה: כל שימוש אחרי הראשון דורש רק את השורה האחרונה

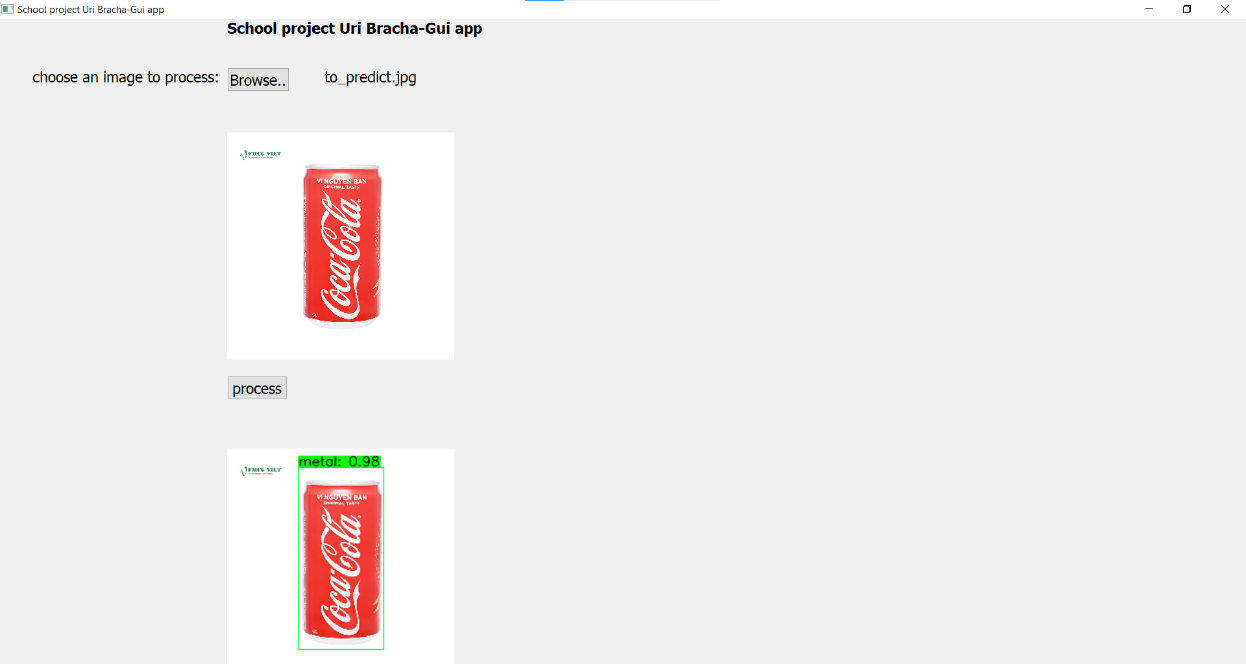
מדריך שימוש בתוכנה

התוכנה נראית כך:



על מנת להכניס תמונה צריכים ללחוץ על כפתור ה- browse, לאחר מכן יפתח חלון שבו אפשר לבחור קובץ תמונה וללחוץ ok. בנקודה זו רואים את התמונה מתחת לקובץ ה- browse ואת שם הקובץ של התמונה. לאחר מכן יש ללחוץ על process ולחכות זמן מה (חצי דקה עד מספר דקות) ואז מתחת ל- process תופיע התמונה עם התיבות התוחמות. אם רוצים לקבל את התמונה כקובץ יש ללכת לתקיית saves שם תופיע התמונה.

תמונה של ה- GUI אחרי זיהוי:



# רפלקציה

פרויקט זה החל כרעיון כאשר עשיתי את עבודת הקיץ, שהיתה בנושא זיהוי פסולת. פרויקט זה הוא שדרוג של הפרויקט המקורי, שבו יכולתי לזהות רק פריט אחד ואחת השאלות ששאלתי את עצמי היא האם אפשר לזהות מספר פריטים.

העבודה על הפרויקט הייתה מעניינת ומסובכת, בהתחלה התקשיתי בעריכת מסד הנתונים TACO ובמציאת הדרך הטובה ביותר להשתמש ב- YOLO. לדעתי רוב הבעיות בפרויקט נגרמו מכיוון ש- TACO לא התאים לעבודה מסוג זה, שבה רוב התמונות כללו בעיקר רקע ומעט מקום לפריטי הפסולת, בעיקר לאחר שהקטנתי את התמונות ל 256\*256.

הקשיים שלי עם YOLO היו בעיקר להבין את מבנה הרשת וכיצד להשתמש בה בצורה נאותה ולאחר שלמדתי את YOLO הבנתי שהבעיות בפרויקט היו בקובץ המידע שלי.

העבודה הייתה מתגמלת מאד ולמדתי ממנה רבות. למדתי על סוגי רשתות כגון YOLO והגרסאות השונות שלה, למדתי כיצד יוצרים ממשק משתמש ב- Python, דבר שאני בטוח שאשתמש בו בעתיד, גם ללא קשר ל- Deep learning ולמדתי כיצד עושים סדר בתוך ה- Google drive, בעיקר במקרה של אימונים ב- YOLO.

אם הייתי מתחיל מחדש את העבודה היום, הייתי עושה מספר דברים אחרת. ראשית, הייתי מחפש מסד נתונים מתאים יותר או יוצר אחד בעצמי מלכתחילה. בנוסף, לאור הידע שיש לי עכשיו ב- YOLO, ככל הנראה הייתי יכול לעבוד עם TensorFlow מלכתחילה, דבר שהיה מקל עלי בנושא ממשק המשתמש. עם זאת, יתכן שהייתי עובד עם Darknet, מכיוון שהאימון ויצירת הגרפים בה הייתה קלה מאד.

המסקנה העיקרית שלי מהפרויקט היא שהנושא של ל- Deep learning מעניין אותי מאד. דרך החשיבה ב- Deep learning שונה מדרך החשיבה של תכנות "סטנדרטי" ואני יכול לראות את עצמי לומד ועובד בתחום זה בעתיד.

דברים שהייתי מעוניין לפתח בעתיד בפרויקט:

* יכולת לעבוד עם קבצי וידאו
* יכולת לקבל וידאו ממצלמה (כגון מצלמת מחשב)
* הגדלת מסד הנתונים לצורך זיהוי טוב יותר, אולי באמצעות שימוש ב- Data agumation.

# ביבליוגרפיה

Techzizou. (2021, October 7). *YOLOv4 vs YOLOv4-Tiny*. Medium. Retrieved February 23, 2022, from https://medium.com/analytics-vidhya/YOLOv4-vs-YOLOv4-tiny-97932b6ec8ec

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016, May 9). *You only look once: Unified, real-time object detection*. arXiv.org. Retrieved February 23, 2022, from https://arxiv.org/abs/1506.02640

Yu, T. Q. (2021, October 8). *Coco to YOLO annotations*. Medium. Retrieved February 23, 2022, from https://medium.com/@thamqianyu96/coco-to-YOLO-annotations-9d638bb3eb4f

hunglc007. (n.d.). *HUNGLC007/TensorFlow-YOLOv4-tflite: YOLOv4, YOLOv4-tiny, YOLOv3, YOLOv3-tiny implemented in TensorFlow 2.0, Android. Convert YOLO V4 Weights TensorFlow, tensorrt and tflite*. GitHub. Retrieved February 23, 2022, from https://github.com/hunglc007/TensorFlow-YOLOv4-tflite