סיווג פסולת באמצעות למידת מכונה-פרויקט סופי מדעי המחשב

שם תלמיד: אורי ברכה

שם בית ספר: תיכונט

תעודת זהות: 325742682

שם המנחה: גד לידרור

שם החלופה: למידת מכונה

תאריך הגשה:



# תוכן עניינים:

[**תוכן עניינים:**](#_kttd1ccxd2te) **2**

[**מבוא:**](#_r8u1y2nxnvuj) **3**

[**הסבר על מסד נתונים**](#_7w3tf23xnzi5) **4**

[yolo ומסדי נתונים](#_pfkven4clyfp) 4

[הסבר על מסד הנתונים TACO](#_wlli1cnicwt2) 4

[שינוי התוויות של טאקו:](#_jdgevdeko03l) 5

[עריכת קובץ הannotation בשביל שיהיה יותר קריא:](#_1ybwxd492tlz) 5

[שינוי גודל התמונה וגודל הannotation](#_41jmcj5x9m7c) 6

[שינוי הannotation מפורמט coco לפורמט yolo](#_c6ub2qwhd7gp) 7

[מסד נתונים שני: מסד נתונים נבנה אישית](#_lmn8y23xmpim) 10

[הסבר על תוכנות תיוג:](#_cie6e74s4j4) 10

[בניית מסד הנתונים:](#_eun52s2t0s0a) 11

[זכוכית](#_8d0blblhvcfv) 11

[מתכת](#_ep6f8vu72kk) 12

[פלסטיק](#_pjpeoxbvwat3) 13

[נייר:](#_vno96q6xfrw6) 15

[קרטון](#_onwjg6hinwjr) 16

[מתכלה:](#_ufw3l1fut6nq) 17

[**הסבר על yolo**](#_5skvkioxi9h1) **18**

[**אימון וחקירת היפרפרמטרים**](#_yfbgvx1upxff) **20**

[הקדמה על אימון](#_2isnytl0r26m) 20

[הסבר על האימון](#_k1xlgms9br20) 21

[אימונים](#_3po3kagz06se) 21

[אימון ראשון](#_xw4cxky5s3ds) 21

[אימון שני:](#_9a77g4okjvdf) 23

[אימון שלישי:](#_qdjr4es7qm8t) 25

[האימון הרביעי](#_v1pxxjtibu2s) 27

[אימון 5](#_otvu9m32tozy) 28

[אימון שישי](#_ryj032g8c7ka) 30

[אימון שביעי](#_2v584qeene36) 32

[אימון שמיני](#_fm4czogjo48m) 34

[אימון תשע](#_nwou08i8ft3h) 35

[סיכום אימונים:](#_cqbd1x6ypkox) 36

[תיאור יצירת הישום:](#_l98n60qztws7) 37

[מדריך למפתח](#_wtqnlx53bauk) 39

[**מדריך למשתמש**](#_fjktrx9rw3yu) **44**

[**רפלקציה**](#_xot3cgz64uxn) **46**

# מבוא:

כמות הפסולת שנזרקת היום היא בעיה משמעותית מאוד. יש לה השפעה רחבה על שינוי אקלים. בישראל בלבד נזרקה 13 מיליון טונות של פסולת ב2019 . אחת הדרכים להפחית את ההשפעה של זיהום שנגרם מפסולת היא מיחזור. לפי ויקפדיה מיחזור הוא תהליך שבו מפרקים פסולת למצב של חומר גלם ומשתמשים בחומרי הגלם האלה כדי ליצור מוצרים חדשים, כתוצאה מכך מיחזור מפחית את הצורך של כריית חומרי גלם חדשים. בנוסף מיחזור עוזר להפחית זיהום אוויר ומאט שינוי אקלים.

בישראל (וברוב העולם). מחזור נעשה באמצעות פחי מיחזור. בעולם בכללי יש שישה סוגים של פחי מחזור: מתכת, פלסטיק, נייר, זכוכית, קרטון(אריזות) ופח לפסולת אורגנית. בישראל הם גם קיימים אבל לא כולם נמצאים באיזורים מספיק קרובים למקומות דיור בשביל שיוכלו להשתמש בהם באופן סדיר. לאחר מכן הפחים נלקחים למתקנים מיוחדים שבו מפרידים אותם שוב בשביל לוודא שלא זרקו לפח הלא נכון ואז מבצעים את תהליך המיחזור.

שבחרתי את הנושא מכיוון שכפי שנאמר הוא נושא משמעותי ובגלל שהתחום של סביבה מאוד חשוב לי וכבר ביצעתי עבודה דומה של למידת מכונה בעבר.

המטרה שלי בפרוייקט הזה הוא ליצור אפליקציה שיכולה לנתח תמונה שיש בה פסולת, ועל כול פריט פסולת שיש בתמונה יסמן אותו ויגיד לאיזה פח צריך לזרוק את פריט הפסולת. קהל היעד במקרה שלי הוא הציבור הרחב, בעיקר אלה שרוצים למחזר ולא בטוחים לאיזה פח לזרוק את הפסולת שלהם.

למיטב ידיעתי אין אפליקציה שעושה בדיוק את מה שאני רוצה לעשות. יש לא מעט אפליקציות שנותנות מידע על הנושא של מיחזור, מיקומים של פחי מיחזור ואפילו היכולת להזמין משאית שתיפטר מהזבל של הבניין. אפילו מצאתי אפליקציה שמאפשרת לסרוק פריט אחד ולקבל הסבר לאן זורקים אותו בשם junker, אבל אין שום אפלקצייה שעובדת על מספר חפצים בו זמנית.

בשביל זה אני צריך להשתמש בסוג מאוד מסויים של מודלים בלמידת מכונה. הסוג שאני מחפשים הם מודלים של object detection. בתחום הזה של המודלים יש שתי ארקיטורות SSD וYOLO ולכל אחד מהם יתרונות וחסרונות. היתרון של SSD הוא העבודה שהוא יותר מהיר בזיהוי התמונות עצמן, מכיון שהרשת שלו פחות גדולה. החיסרון של SSD היא העובדה שיש לו בעיה לזהות חפצים קטנים, yolo לעומת זאת מתאמן יותר מהר יכול לזהות חפצים קטנים יותר טוב ומצאתי עליו יותר מידע באינטרנט.

בגלל שאני יודע שישנה האפשרות שאצטרך לזהות חפצים קטנים, ובגלל שהיה לי זמן מוגבל למחקר בחרתי את yolo. הסבר מורחב על yolo יבוא בהמשך.

היו לי בעבודה שני אתגרים מרכזיים, הראשון הוא למצוא מסד נתונים שמתאים לעבודה שאני רוצה לעשות, בעיקר מסד נתונים שבו יש את הנתונים שyolo דורש. האתגר השני העיקרי הוא ללמוד את yolo עצמו. ולהתגבר על כל הבעיות שייצרו בפרוייקט. הסבר על מסד הנתונים ועל yolo יבוא בפרקים הבאים.

# הסבר על מסד נתונים

## yolo ומסדי נתונים

מודל yolo ומודלי קלסיפיקציה בכללי עובדים בכך שהם לוקחים מסד נתונים, במקרה שלי תמונות מתויגות ולומדים מהם איך לזהות את תמונות חדשות שהם לא אומנו עליהם. מסיבה זאת חלק משמעותי מאוד מעבודה זאת היא מציאת ועיבוד מסד נתונים בשביל שיתאים למטרות העבודה.

בשביל עבודה זו השתמשתי בשני מסדי נתונים, הראשון שניסיתי להשתמש בו הוא מסד נתונים ששמו TACO. ראשי תיבות של פסולת מסומנת בהקשר. מסד נתונים זה עשה בעיות(הסבר בהמשך) ולכן עברתי בניתי מסד נתונים בעצמי. הסבר על כל אחד ממסדי הנתונים יבוא בפסקה נפרדת.

## הסבר על מסד הנתונים TACO

מסד הנתונים טאקו הוא מסד נתונים שנאסף מעבודה של אנשים פרטיים. למסד נתונים זה היו שני חלקים. החלק הראשון הוא 1500 תמונות של פסולת בכל מיני תנאים. החלק השני הוא annotation (annotations). של תיבות תוחמות לכל אחת מהתמונות האלה, שמסמנות את מיקום הפסולת בתמונה.

המידע הזה מאפשר,לכאורה, שימוש במסד הנתונים הזה לאימון yolo יחסית בקלות. לעומת זאת היה צורך בביצוע preproccesing לפני השימוש במסד הנתונים הזה בשביל yolo. בשביל השימוש היה צורך בארבעה שינויים עיקריים:

שינוי הקטגוריות של הannotation בטאקו לקטגוריות הרצויות

עריכת קובץ הannotation בשביל שיהיה יותר קריא (מחיקת מידע לא נחוץ).

שינוי גודל התמונות לגודל אחיד

שינוי גודל הannotation בשביל שיתאימו לגודל התמונה.

### שינוי התוויות של טאקו:

הannotation של טאקו ניתנו בפורמט ששמו CACO. שחלק ממנו אומר הקטגוריות שבו התחלקו לקטגוריות ולקטוגריות-על, החלטתי לשנות את הקטגוריות-על ולהשתמש בהם בשביל הקטגוריות לyolo לעתיד בקובץ הannotation כל קטגורייה נראתה כך:

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

וכל סופר קטגורייה נראית כך:

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

ובקובץ כל אחת מהן בתוך ליסט (מערך) שמאגד אותן יחד.

ולכן בקוד שכתבתי עשיתי את הדברים הבאים:

* טענתי את קובץ הjson שמכיל את המידע של הannotation לשלושה ליסטים שונים: אחת לתמונות, אחת לסימנים, ואחת לקטגוריות ולסופר קטגוריות.
* יצרתי שבע רשימות שונות (לכל אחת מהתוויות שאני רוצה). כל אחת מהן מכילה את המספר הסידורי של הסופר קטגוריות מתאימות לאותה תווית
* עברתי על החלק של הסופר קטגוריות בליסט בשימוש ב for loop
* בתוך הלולאה בדקתי על כל סופר קטגורייה לאיזה תווית היא שייכת (בשימוש במספר הסידורי שלה), ושיניתי את השם שלה לאותה תווית.
* עברתי על כל אחד מהקטגוריות הרגילות שימוש בלולאת for ושיניתי את הסופר קטגורייה שלה לפי הקטגורייה שהייה לה קודם.
* בסוף כל הקוד שמרתי את המידע החדש בקובץ json חדש בשם annotation\_mine.json בשביל שאהיה לי גיבוי של הקובץ הקודם למקרה שמשהו יתשבש.

### עריכת קובץ הannotation בשביל שיהיה יותר קריא:

לפני שינוי גודל התמונות לגודל קבוע. רציתי שקובץ ה annotation יהיה יותר קומפקטי ויותר קל לקריאה בשביל שלא אעשה טעוית. הקובץ היה מורכב מחמישה חלקים"

* מידע: מידע על שנת ייצור הקובץ, מי ייצר אותו וכדומה. את המידע הזה יכולתי להוריד בבטחה.
* תמונות: מידע על התמונות, על כל תמונה היה המיקום שלה (file path). אורך ורוחב, ובאיזה כתובת אפשר למצוא אותן מבחינת url. מידע על מתי הוסף (שלרוב הייה ריק). יכולתי להוריד את כתובת ה url ואת המידע על מתי הוסף.
* annotation: מידע על הannotation, כל סימון הכיל מידע על segmentation שלא הייתי צריך, תיבות תוחמות (שאותם הייתי צריך), שטח (שיכולתי לחשב אותו באמצעות מידע על התמונות התוחמות אז לא הייתי צריך אותו), קטוגוריית הסימון,ומי הוסיף את הסימון.
* קטגוריות: הקטגוריות התחלקו לשניים, קטגוריות רקע, המתארות את הרקע וקטגוריות של החפצים עצמן. הייתי צריך רק את הקטגוריות של החפצים עצמם.

אז בסך הכל הייתי צריך את המידע הבא: על התמונות מיקום קובץ גודל ורוחב, על הannotation: מספר סידורי, תיבה תוחמת וקטגוריה ועל הקטגוריות אני צריך מידע של כל הקטגוריות וסופר הקטגוריות.

בשביל זה כתבתי קוד שעושה את הדבר הבאים:

* טוען את הקובץ של הannotation שיצרתי בקוד הקודם( annotation.json). לשלושה ליסטים: אחד בשביל תמונות השני בשביל הannotation והשלישי בשביל הקטגוריות.
* הכנסתי את החלקים שאותם אני צריך למשתנים נפרדים והוספתי אותם ל dictonary
* השתמשתי בספריית Json בשביל ליצור קובץ חדש בשם annotation\_mine\_short.json.

### שינוי גודל התמונה וגודל הannotation

התמונות של טאקו הן בגדלים שונים, הגדולה ביותר שמצאתי הייתה בגודל 5000\*4000 פיקסלים. בשביל לבצע אימון דרך yolo אני צריך תמונות בגודל אחיד ובגודל מתחלק ב 32 ללא שארית. מסיבה זו הייה צורך לשנות את גודל התמונות.

yolo אומן על מסד נתונים עם תמונות בגודל 416\*416, אבל אני בחרתי להשתמש ב 256\*256. יש לזה שתי סיבות עיקריות, הראשונה היא בעיית הזיכרון: לא הייתי בטוח כמה מקום יש לגוגל קולב בgpu שלהם. ובגלל שיש לי 1500 תמונות לא רציתי להסתכן בשגיאה של אין זיכרון.

ש

הסיבה השנייה היא מהירות האימון, ככל שהתמונות גדולות יותר ככה ייקח יותר זמן לאמן אותם. בנוסף yolo היא רשת גדולה עם הרבה פרמטרים והרבה שכבות. לכן העדפתי לתת תמונות יותר קטנות ולא להקשות את אימון של המכונה, בעיקר כי אני רוצה לעשות מספר סבבי אימונים.

בנוסף לשינוי גודל התמונות חייבים לשנות את הannotation בשביל שיתאימו לתמונות החדשות, ליתר דיוק צריך להכפיל כל אחד מהannotation במספר שנוצר מהמשוואה הבאה:

הגודל החדש של התמונה (256) . חלקי רוחב או אורך הישן של התמונה (תלוי אם מדובר בx או ב y של הסימון).

בשביל זה כתבתי קוד שמבצע את הדברים הבאים:

* טוען את התמונות ואת הannotation למערכים נפרדים.
* יוצר מערך נאמפיי בצורה של (1500,2) שיכיל את המספרים שיוצאים מהמשוואה שכתבתי למעלה
* מכניס את התוצאה של המשוואה למעלה על כל תמונה.

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

* משתמש בספריית PIL כדי לפתוח ולשנות את גודל התמונות ל256 על 256. ומשנה את הערך של האורך והרוחב ל256 במערך של התמונות
* שומר את כל התמונות בתקייה חדשה ששמה images ,נותן לכל אחד מהם שם עםמספר סידורי ומשנה את הערך של מיקום הקובץ במערך של התמונות
* שומר את המידע החדש בקובץ json חדש בשם annotation\_resized.json.

אחרי שהרצתי את כל הקוד הזה, המידע היה כמעט מוכן לשימוש, כל מה שנשאר לעשות הוא להפוך את הannotation מפורמט coco לפורמט yolo.

### 

### שינוי הannotation מפורמט coco לפורמט yolo

COCO (אובייקטים דומים בהקשר) הוא מסד נתונים מאוד פופולרי, שמשתמשים בו בשביל תחרויות של זיהוי אובייקטים בכל העולם(אפילו הגרסאות של yolo נבדקות עליו). לכן מאגרי מידע אחרים נוצרים בפורמט דומה בשביל שיהיה יותר קל להשתמש בהם. מכיוון שידעתי את זה התחלתי לחפש מאגרי קוד שיעזרו בשינוי מ COCO ל yolo. מצאתי קוד שעושה כמעט את מה שהייתי צריך באתר ששמו [medium](https://medium.com/@thamqianyu96/coco-to-yolo-annotations-9d638bb3eb4f).

בשביל להבין איך הקוד עובד צריך קודם להבין את הפורמטים של yolo ו COCO. yolo דורש כל אחד מהannotation שלו להיות בפורמט הבא:

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

כאשר

object-class: מספר מאפס עד שש (במקרה שלי). שמראה לאיזה תווית הסימון שייך.

x\_center: מספר עשרוני(float) שמתאר את המרכז של המלבן התוחם בציר ה x (מנורמל מ0-1).

y\_center:מספר עשרוני(float) שמתאר את המרכז של המלבן התוחם בציר ה y (מנורמל מ0-1).

width: רוחב המלבן התוחם (מספר עשרוני) מנורמל מ0-1.

height: גובה המלבן התוחם (מספר עשרוני) מנורמל מ0-1.

לעומת זאת COCO נותן את הannotation שלו בפורמט הבא:

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

כאשר

category- קטגוריית האובייקט (במקרה שלי התוויות הן בסופר קטגוריות).

image\_id: מספר סידורי של התמונה שבה נמצא הסימון

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, שבאותה נקודה לא הייתי בטוח אם אשתמש בה או בגרסה של yolo בטאנסורפלו דורשת שלכל תמונה יהיה קובץ משל עצמה שמכיל את הannotation שבא

לכן במקרה שלי הייתי צריך ליצור קוד שעושה את הדברים הבאים

* עובר על כל אחת מהתמונות ושולף כל אחד מהannotation שלה.
* על כל סימון, משיג את הסופר קטגורייה שמתאימה לו
* על כל סימון, שולף את ערכי התיבות התוחמות ומשנה אותם לפורמט yolo.
* שומר על כל תמונה קובץ טקסט שמכיל את כל הannotation שלה בשורות נפרדות.

בשביל לעשות זאת יצרתי שלושה פונקציות

הפונקצייה הראשונה נקראה

all\_annot\_for\_image

היא קיבלה כפרמטר מספר סידורי של תמונה, והחזירה את כל הannotation שלו בפורמט מתאים. מה שהיא בעצם עשתה זה שהיא עברה על כל הannotation בדקה איזה מתאימים לתמונה והעבירה עליהם את המניפלוציה שהייה צורך לעשות. ולבסוף מחזירה את כל הannotation של התמונה כמחרוזת.

בשביל לבצע את המניפולציה הזאת יצרתי פונקציה ששמה convert\_annot שנעודה לקחת סימון אחד ולהפוך אותו לפורמט של yolo. הפונקצייה הזאת עושה שלושה דברים.

* לוקחת את הסימון, ומעבירה אותו למצב ביניים שבו התיבה התוחמת שלו נראית כך. (x\_start,y\_start,x\_end,y\_end).
* מעבירה את הסימון החדש דרך פונקצייה שלקחתי מאתר medium שתוארה קודם שמעבירה אותם לפורמט yolo.
* לוקחת את הקטגוריה של הסימון, מגלה את הסופר קטוגרייה שלה.
* בודקת את הסופר קטגורייה שלה כנגד dictonary שיצרתי כדי להפוך אותה למספר.
* בסוף מחברת את הכול ל string לפי פורמט yolo

לפונקצייה מאתר מדיום קוראים convert\_label והיא עובדת כך:

* יוצרת משתנים ל x\_min,x\_max,y\_min ו y\_max לפי המידע שהיא מקבלת מהפרמטרים ו מוודאה שהם בסדר הנכון.
* יוצרת משתנים בשביל הנורמליזציה שהם בעצם 1\גודל התמונה.
* מחשבת את מרכז הריבוע בכך שהיא מחברת את המיקומים ההתחלתיים ואת המיקומים הסופיים ומחלקת בשתיים.
* מגלה אורך ורוחב של הריבוע בכך שהיא מחסירה את המיקום הסופי מהמיקום ההתחלתי של הריבוע.
* מבצעת נורמליזציה (מכפילה את המשתנים שגילתה ב 1\256).
* מחזירה את ארבעת הערכים של התיבה התוחמת בפורמט yolo.

בסוף הקוד אני עובר על כל התמונות, מריץ על כל אחת את הפונקצייה all\_annot\_for\_image ושומר את מה שהיא מחזירה בתור קובץ בשם: image\_ מספר סידורי של התמונה.jpg.

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

לאחר מכן הייתי צריך לנסות אימון, אבל כאשר נסיתי אימון נוצרו בעיות. את הבעיות עצמן אתאר בחלק של האימון, אבל את הסיבות אליהן אני אתאר כן. חשוב להבהיר שהסיבות שלי הן רק השערה, ואני לא בטוח אם הם נכונות.

הסיבה הראשונה שאני חושב שיכלה ליצור בעיות באימון היא העובדה שייתכן שחלק מהאורך והרוחב של התיבות התוחמות קטנות מדי, yolo גרסה שלוש יכול לזהות עד גודל 13\*13 אבל כאשר הסתכלתי על התמונות שלי ראיתי תיבות תוחמות קטנות יותר. לעומת זאת אני לא חושב שזה הגורם.

הסיבה השנייה שאני חושב שיכלה ליצור בעיות באימון היא האפשרות שיש טעות באחד מחלקי הקוד שמשנים את מסד הנתונים, אם יש אז זה ככל הנראה במהלך או אחרי שינוי גודל התמונות, אפשרות אחת היא בקוד שינוי הפורמט של COCO לyolo. לעומת זאת אפילו לאחר שבחנתי את הקוד מספר פעמים לא מצאתי שגיאה.

מכיוון שלאחר מחקר רב וחקירה של מספר דרכים ליישם את yolo לא מצאתי פיתרון. החלטתי שאין ברירה אחרת מלהחליף את מסד הנתונים במסד נתונים חדש, אחד שאותו אני בונה בעצמי. החלק הבא של הכתיבה יעסוק במסד נתונים זה

## מסד נתונים שני: מסד נתונים נבנה אישית

### 

### הסבר על תוכנות תיוג:

כפי שנאמר, yolo דורש גם תמונות וגם annotation (annotations). במקרה של מסד נתונים צריך לסמן את התמונות באופן ידני, בשביל לעשות זאת בצורה יעילה אפשר להשתמש בתוכנות שמאפשרות לסמן ריבועים בשימוש ב ממשק משתמש פשוט.

בגיטאהב אפשר למצוא מספר תוכנות מהסוג הזה מספר דוגמאות טובות שהסתכלתי עליהן הם:

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

[labelImg](https://github.com/tzutalin/labelImg)-תוכנה פתוחה המשתמש ב gui פשוט בשביל לבצע תיוג של תמונות בלבד.

בסוף בחרתי להשתמש ב labelImg בגלל הסיבות הבאות:

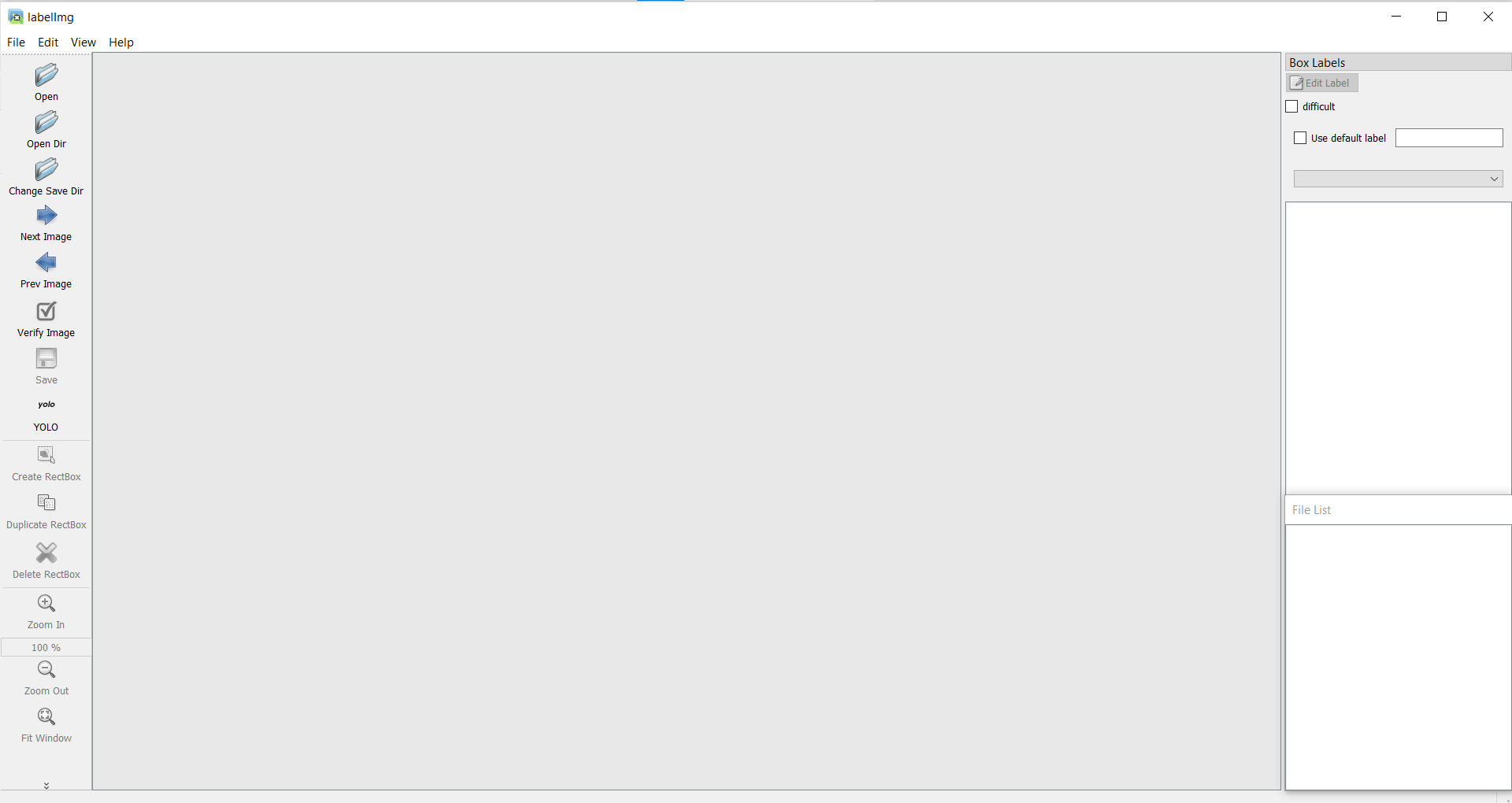
* ה gui הוא פשוט וקל להבנה, מה שמאפשר עבודה מהירה
* יש אפשרות לשימוש בhotkeys שמשפר מהירות עוד יותר
* הכי חשוב, ישנה אפשרות להעביר את הסימנים אוטומטית לפורמט yolo, מה שעוזר להוריד את כמות ה preprocessing ולמנוע טעויות.

את labelimg מריצים בשימוש בפקודה הבאה:

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

Image\_path: המיקום של התמונות

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

ה gui של labelImg נראה כך (תמונה מגיטאהב):

התוכנה עובדת כך:

1. לוחצים על open dir, ופותחים את התקייה עם התמונות
2. לוחצים על open save dir ובוחרים את תקיית שבה רוצים לשמור את קבצי הטקסט של הannotation
3. בכל תמונה לוחצים על create rect או על a בשביל להתחיל סימון
4. מסמנים ריבוע ובוחרים את הclass שלו מרשימת פופ-אפ שמופיעה
5. שומרים
6. חוזרים על 3-5 לכל אחת מהתיבות בתמונה
7. ממשיכים לתמונה הבאה עם d או next image.
8. סוגרים את התוכנה כשמסיימים.

### 

### בניית מסד הנתונים:

לאחר מציאת תוכנה מתאימה לתיוג אפשר להתחיל לעבוד על בניית מסד נתונים. החלטתי להוריד את קטגוריית ה uknown שלי כי כאשר חשבתי על זה הגעתי למסקנה שאם המכונה לא יודעת לאיזה פח צריך לזרוק משהו היא פשוט לא תסמן אותו. לכל קטגוריה אני אכתוב בנפרד מכיוון שתהליך אסיפת המידע היה שונה בכל אחד.

#### זכוכית

את תמונות הזכוכית שלי אספתי באמצעות חיפוש בגוגל. השתמשתי בתוכנה שמאפשרת להוריד את כל התמונות באתר אינטרנט מסויים בשם batch image downloader. החיפוש שהשתמשתי בו הוא glass bottle. והצלחתי להשיג 100 תמונות. כשזה הגיע לסימון היו לי הרבה תמונות עם מספר פריטים שונים בהם. דבר שככל הנראה שיפר את יכולת המכונה לזהות מספר פריטי זכוכית שונים.

למרות שרציתי להשיג 100 תמונות לכל תווית מצאתי שזה לא הספיק בשביל הזיהוי שרציתי, לכן התחלתי להוסיף תמונות. במקרה של זכוכית החלטתי להוסיף תמונות של צלחות וכוסות ומכיוון שהעבודה של סימון היא מסובכת החלטתי להתחיל עם מספר קטן מאוד של תמונות להוספה: 15 צלחות ו 15 כוסות.

את תמונות הזכוכית לקח הכי הרבה זמן לסמו, מכיוון שהייתי חדש לתוכנה שבה אני משתמש לסיווג וגם בגלל שהיו הרבה תמונות מסוגים שונים. תמונת דוגמה לסימון:

#### מתכת

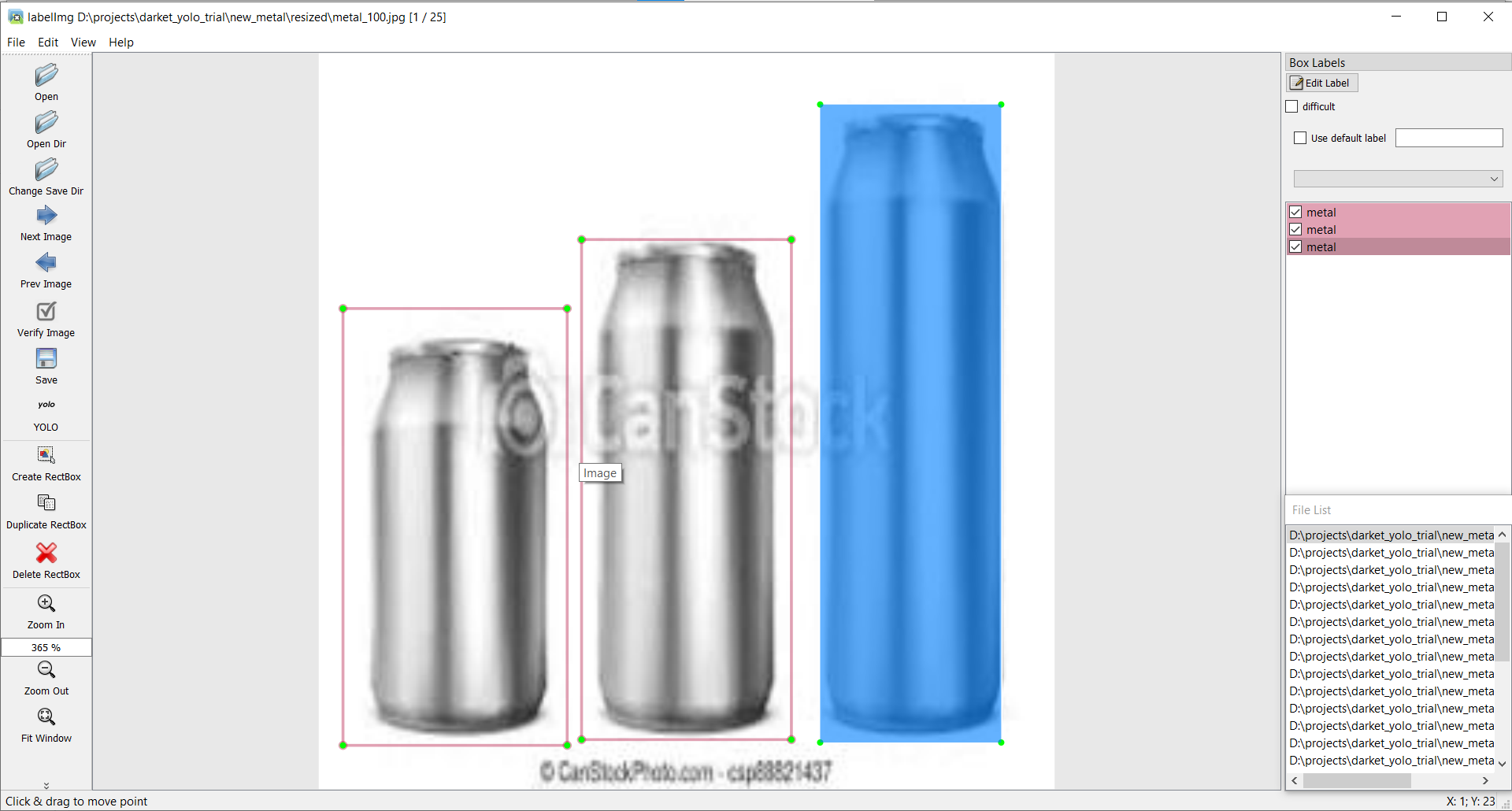
את 100 תמונות המתכת המקוריות שלי מצאתי במסד הנתונים שהשתמשתי בו בשביל עבודת הקיץ בלמידת מכונה, היו בו 500 תמונות ובשביל לבחור 100 תמונות רנדומליות כתבתי קוד של שמתשמש בספריית גלוב (glob). בשביל לשלוף את כל התמונות ובנאמפיי בשביל לבחור מאה מהם בצורה ראנדמולית.



העובדה שבחרתי תמונות ממסד הנתונים משנה שעברה מעניקה יתרונות וחסרונות. היתרונות הם שאני מכיר את התמונות, ויודע בדיוק איך לתייג אותם. החיסרון הוא שהתמונות האלה נבנו בשביל קלסיפיקציה רגילה מה שאומר שיש פריט אחד בכל תמונה והפריט מכסה אחוז גדול יחסית מהתמונה.

לאחר שבחנתי כיצד המודל עובד עם התמונות החדשות. גיליתי שיש בעיות עם זיהוי מספר פריטים (כפי שחזיתי שהייה). אז הוספתי עוד 25 תמונות של בעיקר פחיות שתייה כי הם הדבר הכי משמעותי שזורקים. ובכל תמונה הייה מספר פחיות שתייה, ולאחר מכן כשבדקתי זה בהחלט שיפר את הזיהוי.

תמונה דוגמה לזיהוי:



#### 

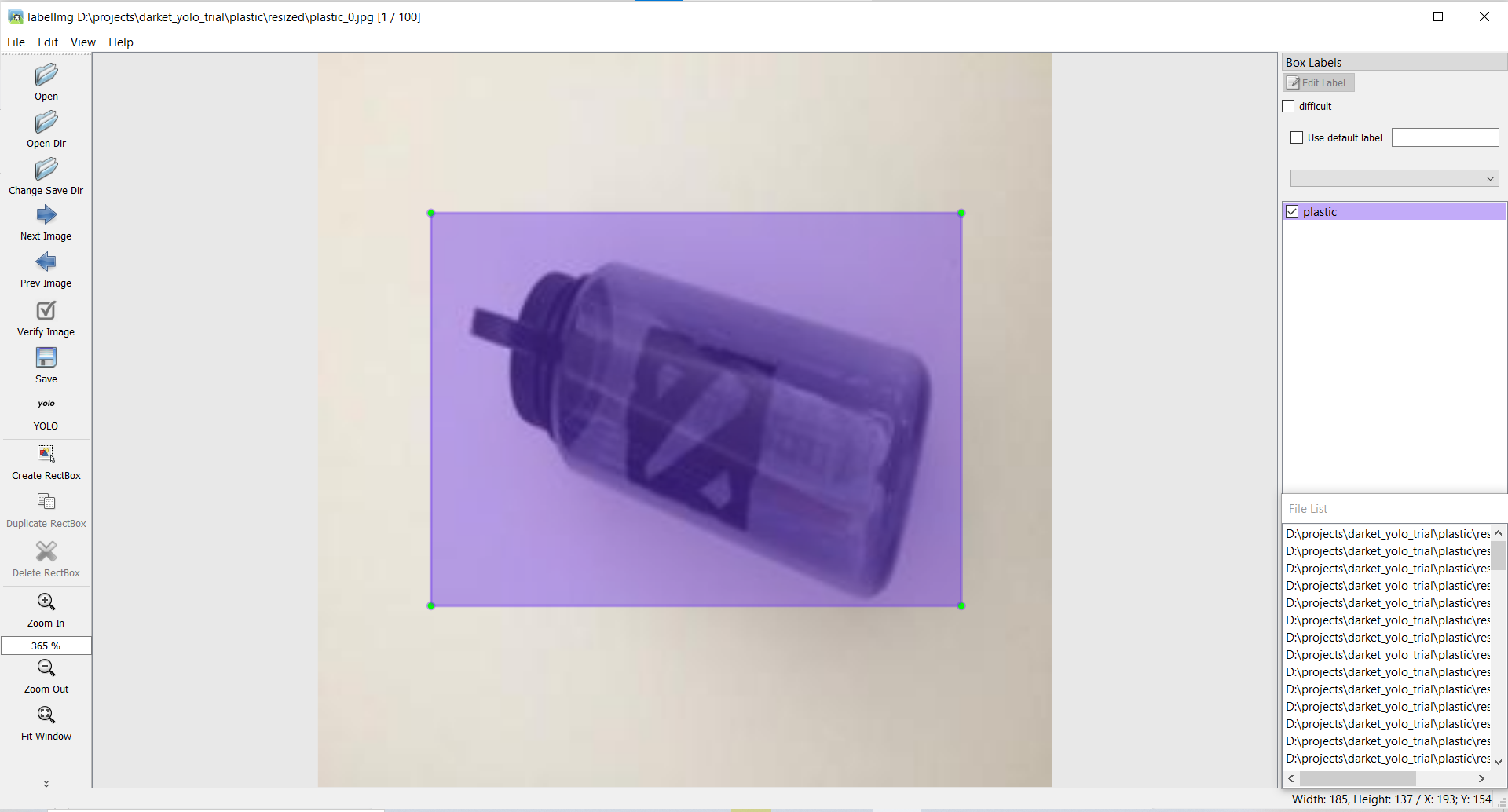
#### פלסטיק

את החלק של התמונות מפלסטיק לקחתי גם ממסד הנתונים של עבודת הקיץ של שנה שעברה (עבודת הקיץ). רוב התמונות היו בקבוקי פלסטיק ופקקי פלאסטיק, בגלל בכמעט בכל התמונות יש פריט אחד, אז כמו הפעם הקודמת חזיתי בעיות פוטנאצליות בזיהוי מספר פריטים.

פלסטיק היא אחת הקטגוריות היותר מסובכות לזיהוי במודל מכיוון שפריטים מאוד שונים נמינים בתוכה. המודל צריך להיות מסוגל לזהות בקבוקים מפלסטיק, קופסאות מפלסטיק, פקקים, ואולי אפילו דברים כמו לגו או ניילון.

עוד סיבה שפלסטיק הינה קטגורייה קשה היא שייתכן שהמודל יתבלבל בין פלסטיק למתכת, בבקבוקים או בפקקים. לכן החלטתי להריץ מספר בדיקות על קופסאות שעושייות מפלסטיק. המודל זיהה אותם נכונה מבחינת המיקום אבל חשב שהם מברזל, לכן הוספתי עוד 25 תמונות של קופסאות פלסטיק בצבעים וזוויות שונות.

תמונה דוגמה לזיהוי:



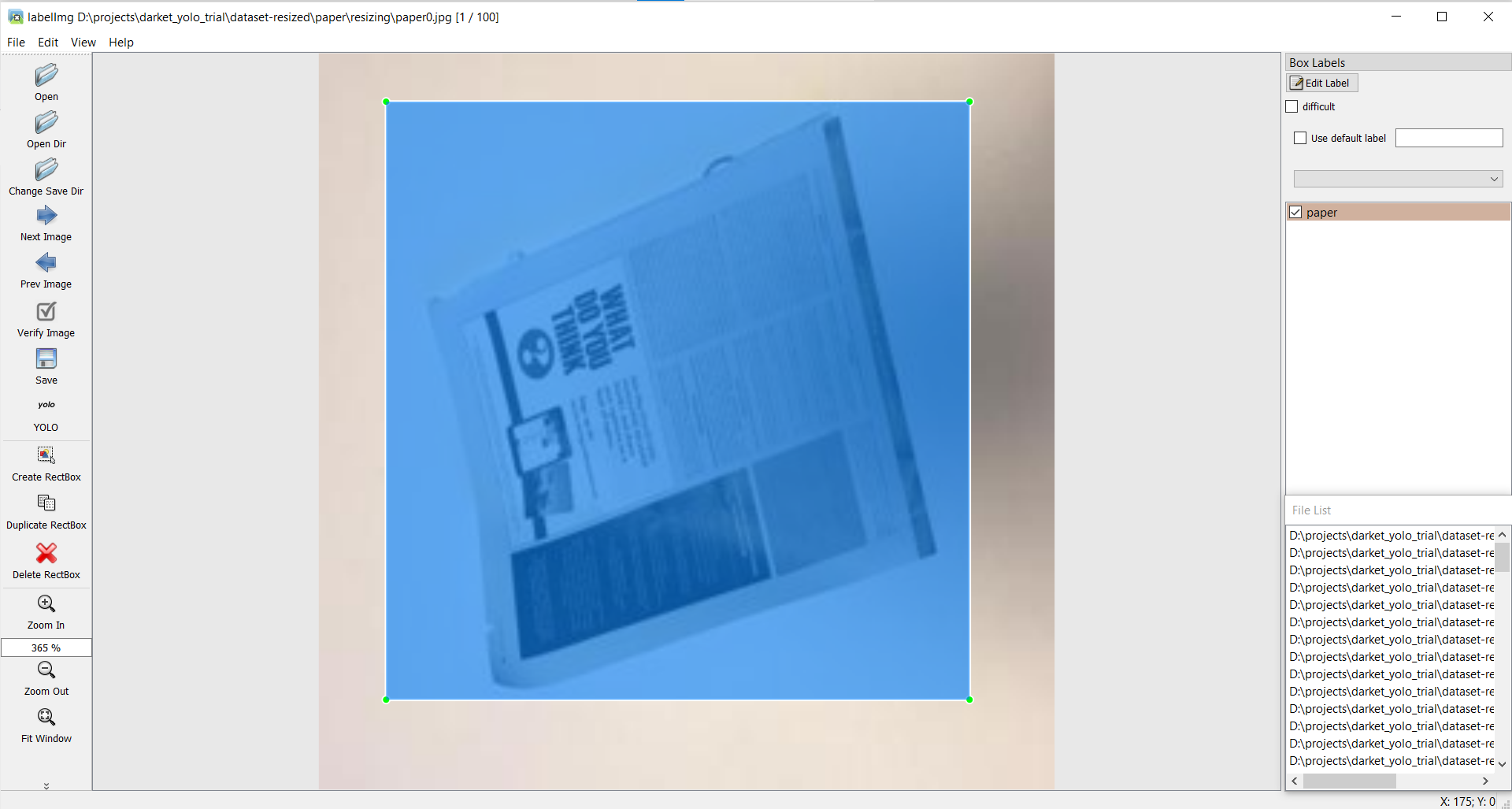
#### נייר:

במקרה של נייר רוב התמונות הם עיתונים אבל ישנם מספר תמונות של מגזינים וספרים. התמונות נלקחו מחיפוש על עיתונים בגוגל ומעורבב בהם כתב מעברית וכתב באנגלית, כמו כל שאר התוויות גם כאן שמתי 100 תמונות בהתחלה.

עם נייר נוצרה בעייה מוזרה במהלך הבדיקה של המודל, אם משתמשים בצילומי מסך מטלפון המודל יוצר תיבה תוחמת שמכילה את כל התמונה עם זיהוי של נייר. מכיון שהעבודה לא משתמשת בצילומי מסך מטלפון אני לא חושב שיש צורך לתקן את הבעייה הזאת.

במקרה של נייר לא הרגשתי צורך להוסיף עוד תמונות מכיוון שהקטגורייה היא יחסית קלה לזיהוי ואין לה הרבה מגוון. אבל אם יהיה צורך להוסיף עוד תמונות אכתוב כאן.

תמונה דוגמה לזיהוי:



#### קרטון

תמונות של קרטון הם בעיקר קופסאות קרטון וסוג של אוריגמי מקרטון. במקרה הזה נסיתי עוד פעם לקחת תמונות באמצעות חיפוש בגוגל של קרטון בשביל לראות אם זה יותר יעיל מתמנות ממסד הנתונים של עבודת הקיץ.

במקרה של תמונות קרטון מצאתי רק 93 תמונות מתאימות (גדולות מספיק, עם פריטים גדולים מספיק). הannotation היו מעט קשים כי הייתי צריך להחליט איפה יש פריט אחד ואיפה יש כמה פריטים, במקרה שיש מספר פריטי קרטון מחוברים יחד.

ישנה אפשרות שהמודל יתבלבל בין קרטון לבין קופסאות פלסטיק אבל אני לא חושב שזה תהיה בעייה מכיון שהצבע של קרטון והצבע של קופסאות פלסטיק שונים מאוד. בגלל זה יש יתרון שיש לי מספיק מקום לשימוש ב rgb ושאני לא חייב להשתמש ב greyscale כמו בעבודת הקיץ שעשיתי.

תמונה דוגמה לזיהוי:

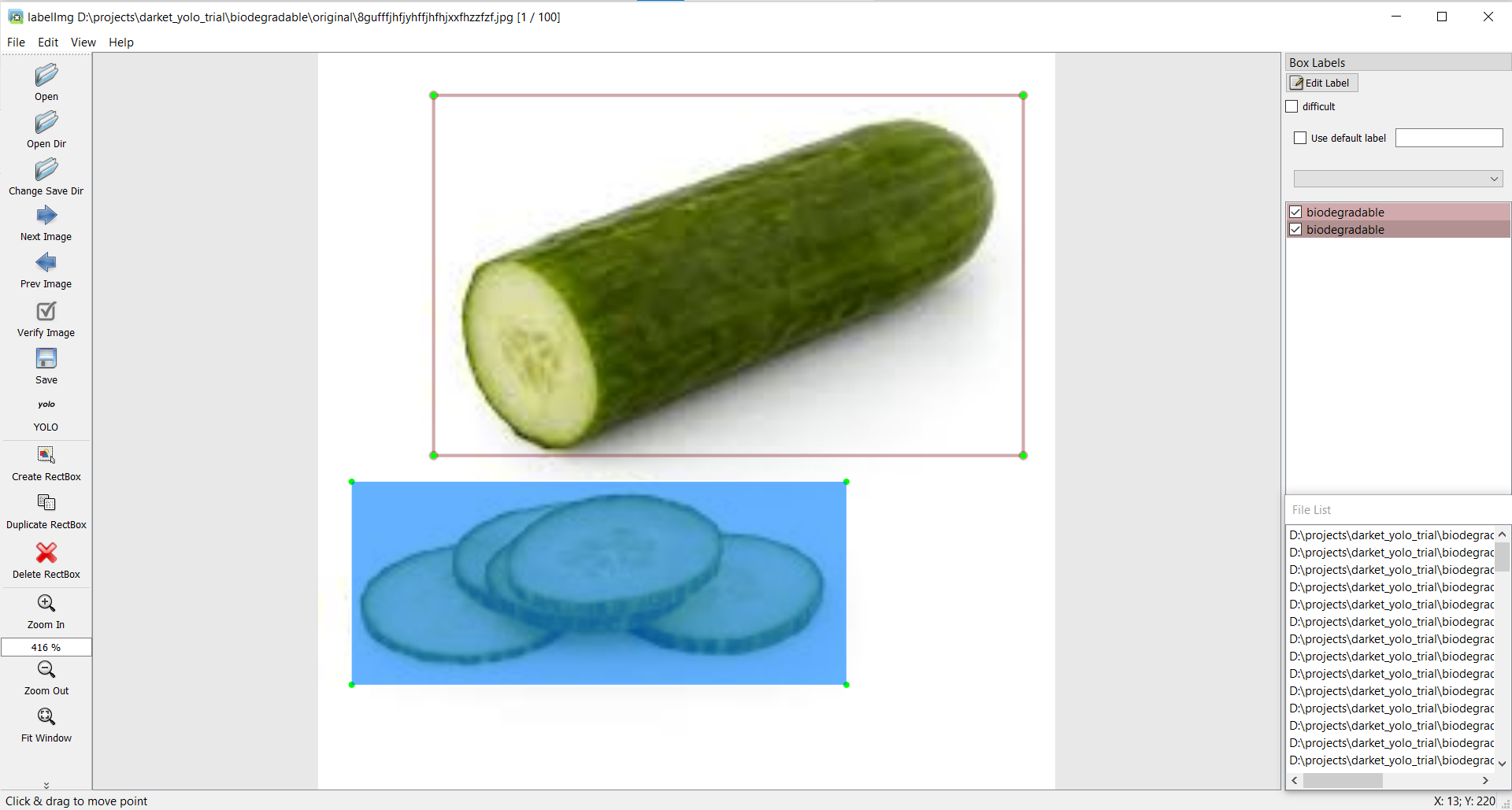


#### מתכלה:

מתכלה (biodegradable) זה בעיקר תמונות של ירקות, פירות, אגוזים קליפות וכדומה. במקרה הזה גם לקחתי תמונות מגוגל. את התמונות שלקחתי חילקתי בדרך הבא: 70 תמונות בשביל פירות וירקות ו30 בשביל אגוזים, עוגות ופשטידות. הירקות והפירות שלקחתי היו: תפוח, תפוח אדמה, בצל, מלפפון, עגבנייה, תפוז ובננה.

במתכלה אני לא הייתי מודאג שהיו קונפליטים עם תוויות אחרות, כי אין כמעט שום דמיון בין זה לבין הקטגוריות האחרות. ובבדיקה הראשונית שלי זה באמת הייתה הקטגורייה שהמודל היה הכי בטוח לגביה. עדיין לא הספקתי לבדוק על שילובים של מספר פירות יחד אבל אני חושב שגם בזה לא תהיה בעייה.

תמונה של סימון:



# הסבר על yolo

בדרך הסטנדרטית של זיהוי אובייקטים (רשת למידה עמוקה שנגמרת עם softmax). אפשר לזהות רק אובייקט אחד שהוא ככל הנראה האובייקט היחיד בתמונה, ואין דרך לדעת את מיקומי של האובייקט. במקרה של העבודה שלי אני צריך לזהות מספר אובייקטים שיכולים להיות בעלי תוויות שונות. במקרה הזה שיטת למידת מכונה ששמה yolo נכנסת לתמונה .

yolo , ראשי תיבות של הסתכלות רק פעם אחת הוצעה ב 2015 במאמר [You Only Look Once: Unified, Real-Time Object Detection - arXiv](https://arxiv.org/abs/1506.02640). הרעיון הבסיסי של yolo הוא פשוט: מחלקים כל תמונה לשריג(grid) בגודל מסויים, ובודקים האם בתוך כל תא יש פריט שמתאים לאחת מהתגיות שלנו. בשביל לעשות זאת לרוב או לכול התמונות במסד הנתונים שלנו צריך להיות תיבה תוחמת (bounding box) על כל אחד מהפריטים שמתאימים לאחת מהתגיות שלנו. (לכן הייתי צריך להשתמש ב תיבות התוחמות ב annotation.json).

שימוש בתיבות תוחמות יוצר שתי בעיות פוטאנצליות. הבעייה הראשונה היא המקרה שבו פריט נמצא בתוך שני תאים או יותר של השריג. במקרה הזה yolo משתמש בשני אלגרויתמים כדי לפתור את הבעיה. הראשון הוא iou, האלגוריתם הזה לוקח שתי קופסאות(תיבות תוחמות) ומחזיר את ה ratio בין בין שטח החיתוך של המלבנים לבין סכום השטחים שלהם.

האלגוריתם השני הוא Non Max Suppression. האלגוריתם מתחיל בכך שהוא זורק את כל המלבנים שלא עברו סף (threshold) מסוים, לאחר מכן האולגריתם לוקח את המלבן עם ההסתברות הכי גבוהה באזור וזורק את כל המלבנים שהחפיפה (iou) שלהם איתו עוברת סף מסוים. האלגוריתם הזה עובד אך ורק במהלך הפעלת הרשת, ולא במהלך האימון.

הבעייה השנייה היא המקרה שבו ישנו יותר מפריט אחד בתא(cell). בשביל זה yolo משתמש בעוגנים (anchors). כל תא מתחלק למספר של עוגנים וכל עוגן יכול לזהות פריט נפרד. לדוגמה אם יש תמונה שבא בתא אחד יש אדם עומד וכלב אז עוגן אחד יכול להיות ארוך וצר(כדי לזהות את האדם) ועוגן שני יכול להיות רחב וקצר (כדי לזהות את הכלב). yolo נותן עוגנים ברירת מחדל, ולכן אפשר להשתמש בהם בשביל אימון, במקרה שתיווצר בעייה אצטרך לחשב את העוגנים מחדש.

פונקציית המחיר של yolo 4 בנויה משלושה חלקים

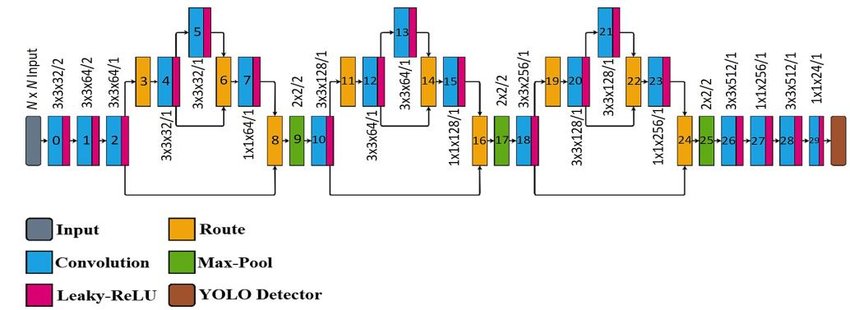
* בינארי קרוס אנטרופי בשביל האם יש אובייקט בכל אחד מהאנקורים, השוואה בין הניחוש לאמת (y true).
* בינארי קרוס אנטרופי בשביל לבדוק איזה אובייקט יש בכל אחד מהאנקורים.
* וחישוב בשימוש ב iou בשביל לחשב את המרחק של התיבה התוחמת שנוחשה מהתיבה התוחמת האמיתית. החליף mean squred error בגרסאות הקודמות.

כפי שנאמר yolo מחלק את התמונה פעמיים פעם ראשונה לסריג שגודל כל אחד מתאיו הוא 32 אז במקרה של תמונה בגודל 256\*256 יש סריג בגודל 8\*8 ואז לעוגנים (במקרה של yolov4-tiny שש כאלה) ולכן כאשר הרשת מחזירה את הניחוש שלה (יאי גג) 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 או לא.

בגלל גודל כל תא בגריד yolo מקבל אך ורק תמונות שהגודל שלהן (גובה ורוחב) מתחלק ב32. לכן היה צורך (הגרסה המקורית משתמש ב 416\*416) אבל במקרה שלי העדפתי להשתמש ב 256\*256 בשביל שהאימון יהיה מהיר יותר.

הגרסה של yolo שבה אני משתמש היא yolo ארבע-קטן, yolo ארבע-קטן בגלל שהאימון שלו הרבה יותר מהיר (לפעמים אפילו פי 30 יותר מהיר). המודל משתמש ב 6 אנקורים במקום 9 ומבצע שני זיהויים בגדלים שונים במקום שלושה. החיסרון שבא אם המהירות הזו הוא דיוק מופחת (⅔ ממה שyolo ארבע יכול להשיג ברוב המקרים). אבל אני מעדיף לאמן את המודל במהירות מספיק, ולא לחכות מספר ימים על כל מודל.

תיאור גרפי של המודל:



בחלקים של המודל שהם Route מה שבעצם נעשה זה לקיחת את המידע שהיא מקבלת, מפצלת את המידע בשניים ומחזירה את החצי הראשון.

החלקים של yolo דאטאקטור זה בעצם שכבת קונבולוציה של 3\*(מספר הקטגוריות+5). שלוש בגלל שלכול זיהוי יש 3 אנקורים ו מספר קטוגריות+5. בגלל שה y\_true ו ה y\_pred של yolo הוא מספר הקטגוריות+5 . חשוב לציין שיש שני זיהויים ובציור מתואר רק אחד.

בנוסף בכל פעם שיש יותר מחץ אחד שמגיע לשכבה מסויימת נעשה לפני זה concatantion שזה בעצם חיבור כל תוצאות השכבות שנכנסות לשכבה.

# אימון וחקירת היפרפרמטרים

## הקדמה על אימון

בשביל ליצור מכונה מסוג yolo שתוכל לזהות פסולות צריך לאמן אותה באמצעות הדוגמאות. בשביל לעשות זאת היו לי כמה אפשרויות. האפשרות הראשונה הייתה לעבוד עם ספרייה שיוצרת מודל yolo באמצעות טאנסורפלו. לאחר חיפוש בגיטאהוב מצאתי מספר מודלים מוגדרים בטאנסרפלו אבל לכול אחד מהם היה בעיות. הראשון שלקחתי עשה בעיות עם ה loss function והיה גם מוגדר עם tensorflow 1.

השני שבחרתי היה מוגדר ב tensorflow 2 אבל היה בעיות עם ה loss function שלא הצלחתי לפתור הבעיות היו קשורות למספר פונקציות שלא היו מוגדרות ועדיין קראו להן. והשלישי שמצאתי עשה בעיות עם הגדרה של הנתונים (x\_train ו y\_train). גם הייה צורך לשנות את הדרך שבה הannotation ניתנים למכונה, מהדרך שבה תואר בחלק של המסד נתונים, לשני קבצים שבהם בכל שורה יש מיקום תמונה וכל הannotation שלה מופרדים ברווחים.

לאחר מכן עברתי לשימוש ב ספרייה ששמה darknet (darknet). שהם היוצרים המקוריים של yolo. בשביל לגרום לזה לעבוד הייתי צריך לעשות עוד קצת preprocessing. דארנט דורש שלכל תמונה יהיו ה annotation שלה בקובץ נפרד. ויהיו שני קבצים על train ו test שיכולו את הpath לכל אחד מהקבצים שבו הם ישתמשו. ויצרתי את הקבצים בשימוש בקוד שלוקח את כל אחת מהתמונות, ובודק אותן נגד ליסט של אינדקסים.

עוד שינוי שהיה צריך לעשות הוא בקובץ הקונפיגורציה של yolo. קובץ הקונפיגורציה הזה קובע את המבנה של הרשת של yolo. לדוגמה הוא קובע איזה שכבות יהיו שכבות קונבצולציה ואילו שכבות יהיו שכבות זיהוי. השינוי הראשון שהיה צריך לעשות הוא לשנות את גודל התמונה של ה אינפוט ל256\*256 אחר כך היה צורך לשנות את שכבות ה[yolo] שהן שכבות הזיהוי של המכונה, בהן צריך לזהות את כמות ה class מהדיפלוט שהוא 80 לשבע. לאחר מכן היה צורך לשנות את שכבות הקונבולציה לפני הזיהוי, בהן הפילטרים היו צריכים להיות (classes+5)\*7.

אחת הבחירות הכי משמעותיות היו איזה גרסה של yolo להשתמש, התחלתי עם yolo 3, מכיון שהוא הכי קל להבנה. אבל לאחר שראיתי שלוקח ל גוגל קולאב הרבה זמן לאמן את המכונה (4 שניות לכל קבוצת תמונות) ובידיעה שאצטרך להריץ הרבה קבוצות החלטתי לעבור לyolo 4 קטן. שהיא גרסה הרבה יותר מהירה(פחות משנייה לקבוצה). ושנותנת תוצאות טובות מספיק ברוב המקרים.

## הסבר על האימון

המטרה העיקרית באימונים זה לבחון מספר קונפיגרציות שונות של הרצת yolo, בעיקר לשנות את גודל הקבוצה, אורך הלמידה(באיטרציות), והכי חשוב שינויים בlearning rate. לפני כל אימון יהיו ההגדרות שהתמשתי בהם בשביל אותו אימון, אחרי כול אימון תהיה פסקת מסקנות, תמונה של גרף ותמונה של זיהוי של המודל(בהם לא בהרכח יהיה הרבה הבדל בין ריצה לריצה).

גרף של אימון של yolo מכיל שני חלקים, החלק הראשון הוא גרף הloss של המודל שמחושב כל מאה איטרציות, החלק השני הוא הדיוק, שמחושב פעם ראשונה אחרי 1000 איטרציות, ואחרי כל 100 איטרציות עוד פעם(הסבר על חישוב הדיוק נמצא בחלק המודל).

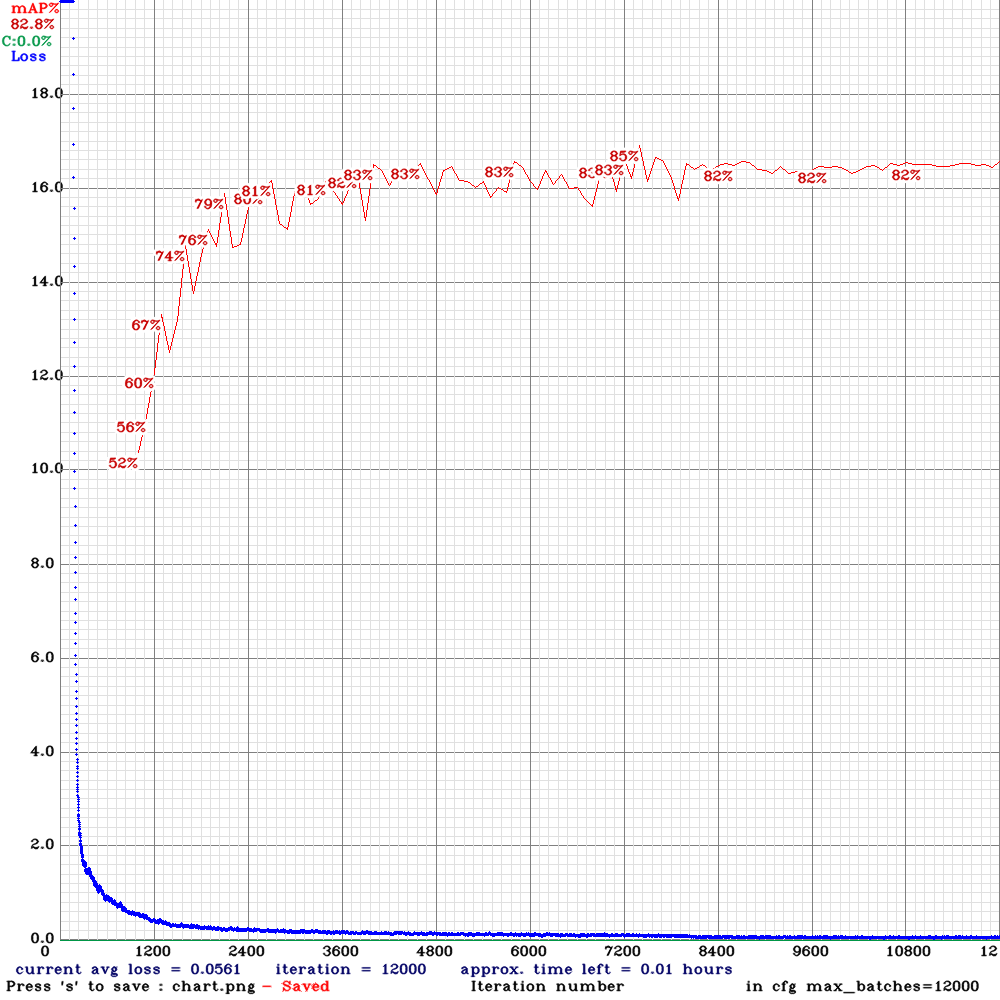
## אימונים

### אימון ראשון

האימון הראשון הוא אימון בשימוש בהגדרות המומלצות של darknet. האימון הזה הוא הקו הבוחן של כול שאר האימונים, מה שאומר שהוא הראשון שאני משווה אליו. הגדרות של האימון:

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

max\_batches זה כמות האיטרציות, batch זה גודל קבוצה ו learning\_rate זה ה . learning rate

גרף:

האימון על פי גוגל קולאב לקח שעה ו 11 דקות. בשביל אחוזי זיהוי כאלה זהו זמן סביר ביותר, בעיקר בהתחשב בכך שגרסה סטנדרטית של yolo (לדוגמה yolo שלוש) הייתה לוקחת בין 9(הערכה שלי) ל20 שעות(מה שyolo אמר).

לפי מה שרואים בגרף המחיר ירד בהתחלה מאוד מהר, כי הוא היה מאוד גבוה הרבה מעל (18) עד שאיפשהו אחרי 200 איטרציות הירידה הפכה להיות יותר מתונה. לעומת זאת הדיוק (MAP) התחיל ב52 וזינק ל79. ואז התחילה מגמה של עליה וירידה שבה המודל הגיעה ל85% ומאז לא הצליח לשפר את עצמוץ

לדעתי המודל בוודאות מאומן טוב מדי על דאטאסאט האימון(overfitting). אבל אחוזי הזיהוי עדיין טובים מאוד, ייתכן שאפשר להוריד את כמות האיטרציות לאימון(אסתכל על זה בהמשך).

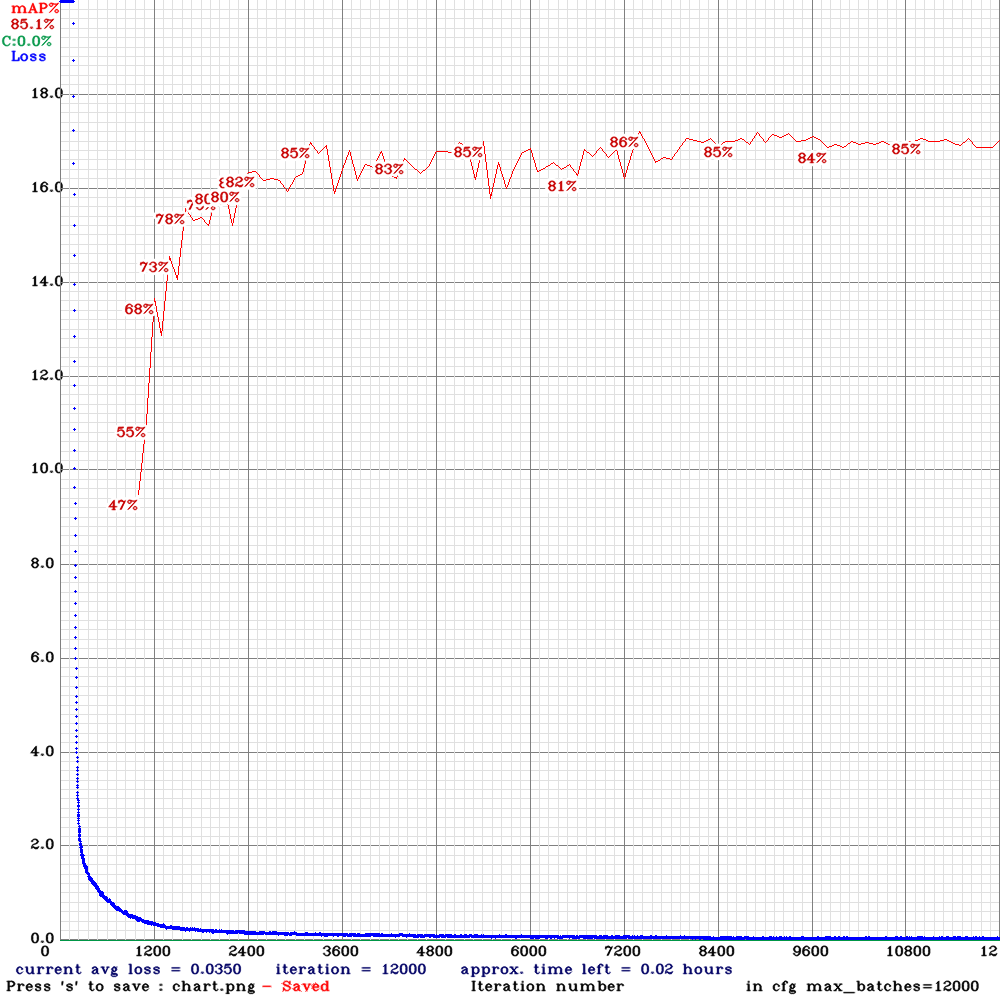
המספרים שכתובים למעלה הם מספרים עשרוניים המעידים על בטיחות המודל בתוצאות שלו. 0-1. 1 זה בטוח לגמרי.

### אימון שני:

שני האימונים הבאים יבדקו את ההשפעה של גודל הקבוצה על האימון של yolo. ההשערה שלי היא שככל שגודל הקבוצה יותר גדול ייקח לyolo יותר זמן לאמן מכיוון שהוא צריך להסתכל על יותר תמונות כל יטרציה, לעומת זאת ייתכן שהמודל יהיה יותר מדוייק. ההגדרות של האימון:

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

גרף של אימון:



כפי שאפשר לראות המחיר ירד יותר נמוך במהלך האימון הזה. אפשר גם לראות שהדיוק הגיע ל86 אחוז בנקודה מסויימת במקום 85 אחוז. כל זה אומר שמבחינת דיוק המודל הנוכחי דומה מאוד למודל הקודם אבל קצת יותר טוב.

לעומת זאת, מבחינת זמן המודל הזה לקח כמעט פי שתיים זמן לאימון. זה לא מפתיע כי כל איטרציה המודל היה צריך ללמוד מפי 2 תמונות (128 במקום 256). לפי דעתי המודל הזה מוכיח שבשימוש במידע הנוכחי 86 אחוז הוא המקסימום של המודל.

אחד הדברים שחייבים להיאמר הוא שהמודל הזה הגיע ל85 אחוזי זיהוי תוך 3000 איטרציות לעומת 7500 במודל המקורי, ייתכן שבדרך הזו אפשר להפוך את המודל למהיר יותר דרך הורדת איטרציות. אבחן את זה בבדיקה עתידית.

לסיכום, הקונפיגורציה של האימון הזה היא פחות יעילה אבל היא פותחת אפשרויות לעתיד, לדוגמה לאמן פחות איטרציות, או אפילו לנסות לשנות את ה learning rate. את האפשרויות נבחן באימונים עתידיים.

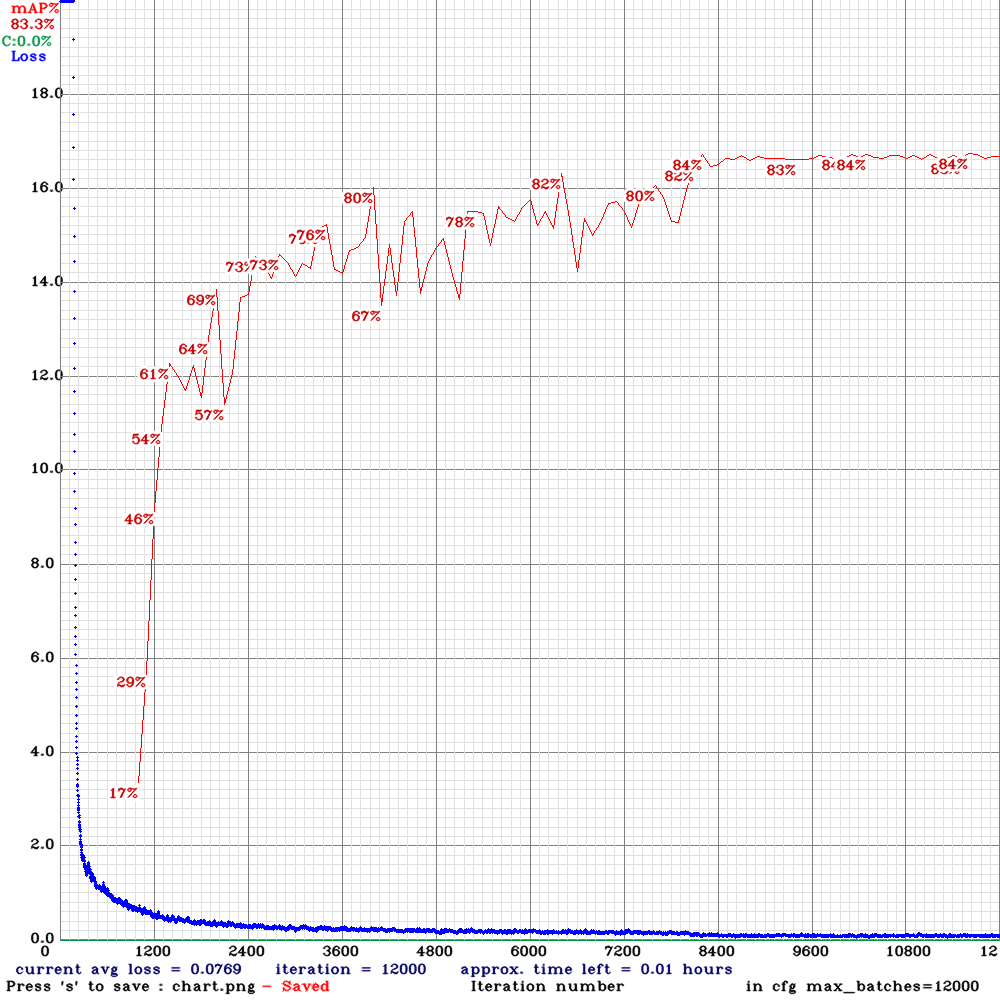
### אימון שלישי:

האימון השלישי יהיה בדיוק ההפך מהאימון השני, נקטין את גודל הקבוצה במקום להגדיל. אני חוזה שהמודל גם יגיב ההפך, עם תוצאות פחות טובות אבל אימון יותר מהיר.

הגדרות של האימון:

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

גרף של אימון:



לפי הגרף אפשר לראות האימון הזה היה פחות מוצלח מהראשון ומהשני מבחינת loss וMAP. האימון היה יותר מהיר מהאימון השני אבל איטי יותר מהאימון הראשו 79 דקות לעומת 119 דקות (מהיר יותר). ו79 לעומת ה 71 (איטי יותר).

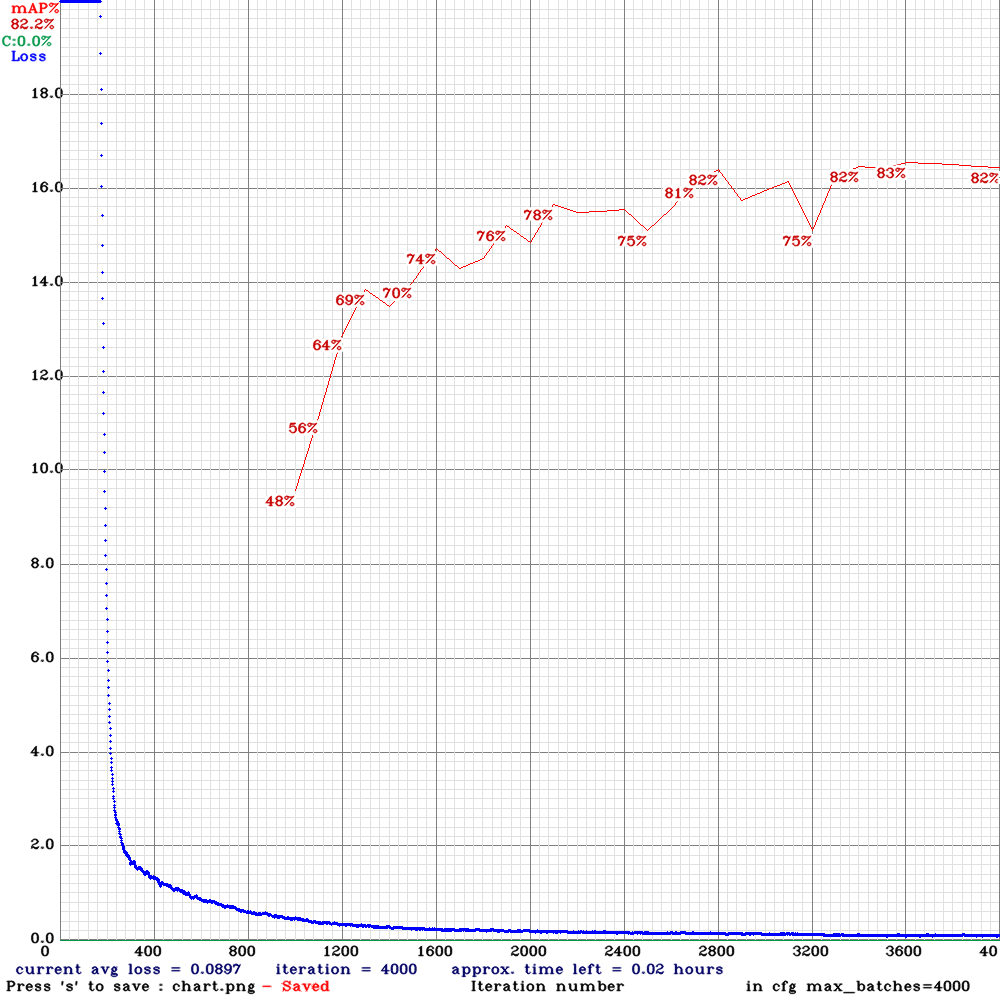
האימון הזה הוא לא טוב במיוחד, הוא גם פחות מדוייק משני האימונים וגם פחות מהיר מאחד. בגלל זה אני חושב שאם הייתי צריך לבחור משלושת האימונים שעשיתי עד עכשיו לא הייתי בוחר את זה.

### האימון הרביעי

האימון הרביעי הוא האימון שכתבתי עליו באימון שתיים, בעצם batch 128 ו 4000 איטרציות. לפי דעתי זה יהיה האימון הכי יעיל בארבעת האימונים, גם מהיר יחסית לאחרים בגלל שעושים רק שליש מהאיטרציות וגם, לפי הגרף של אימון 2 יביא דיוק טוב (85%).

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

גרף של אימון:



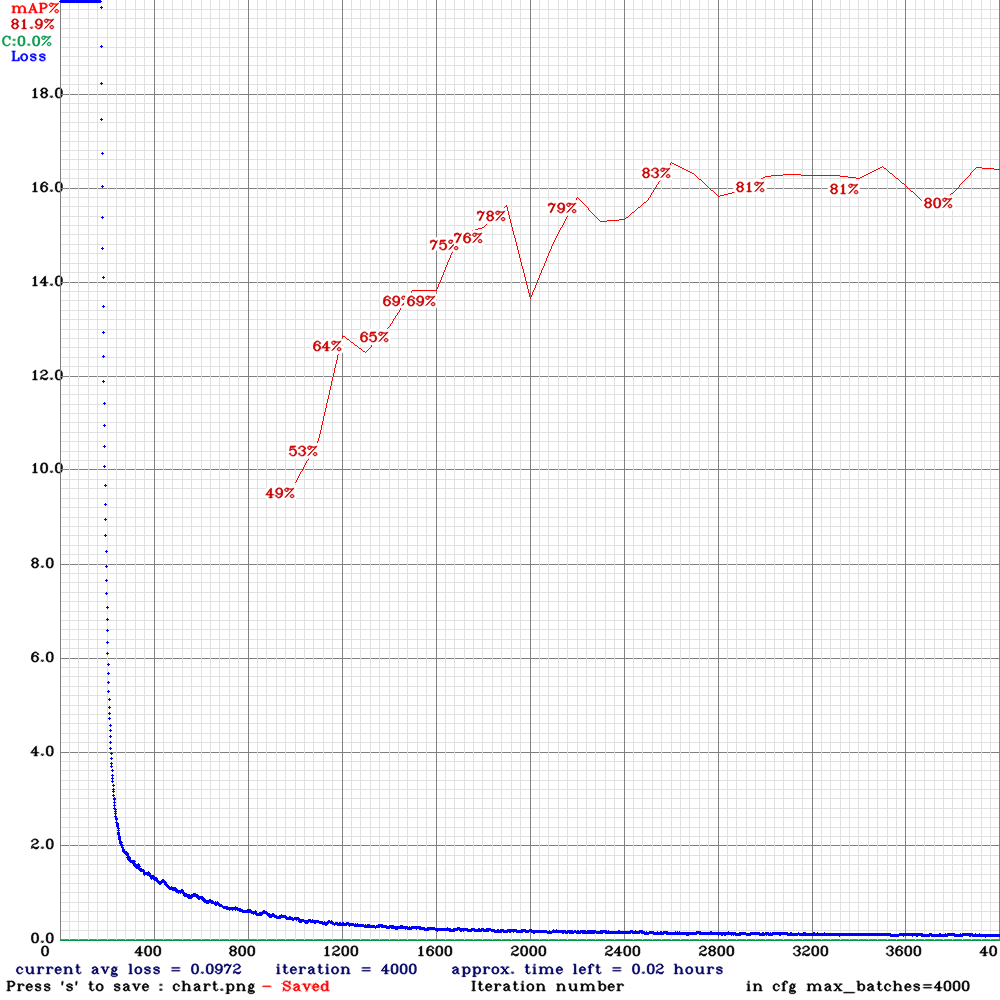
המודל לא היה טוב כמו שציפיתי (82%) במקום 85%, הסיבה לפי דעתי היא בפרמטר שעוד לא הסברתי את קיומו: steps, סטפס הוא פרמטר המתאר לyolo איך לשנות את ה learning rate במהלך הריצה, והוא אמור להיות שתי מספרים, הראשון 80 אחוז מכמות האיטרציות והשני 90 אחוז מכמות האיטרציות. המודל שהצליח להגיע ל85 אחוז ב4000 איטרציות ביצע את זה עם סטפס שהתאימו ל 12000 איטרציות והמודל שהגרף הזה מתואר בו עושה את זה עם סטפס שמתאימים ל 4000 איטרציות.

לעומת זאת המודל לא רע, הוא מצליח להגיע ל 83 אחוז ב47 דקות! המודל המהיר ביותר עד עכשיו, לעומת זאת יכול להיות ששינוי הסטפס יעזור למודל להגיע למה שהוא הגיע. הבחינה של זה תהייה באימון הבא.

### אימון חמישי

כפי שנאמר המטרה של אימון זה היא לבחון האם שינוי של הsteps ישיג את אותן התוצאות כמו במודל מספר 2 ב4000 איטרציות

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

גרף אימון:  


באופן מפתיע ביצוע המודל לא השתפר, המקסימום שהמודל הגיע אליו הוא 83 אחוז. לעומת זאת המודל עדיין הייה מהיר(47 דקות). אבל ייתכן שההבדל של הסטפס הוא לא היחיד מהמודל השני. עדיין המודל נראה יחסית טוב, הוא עובד לא רע.

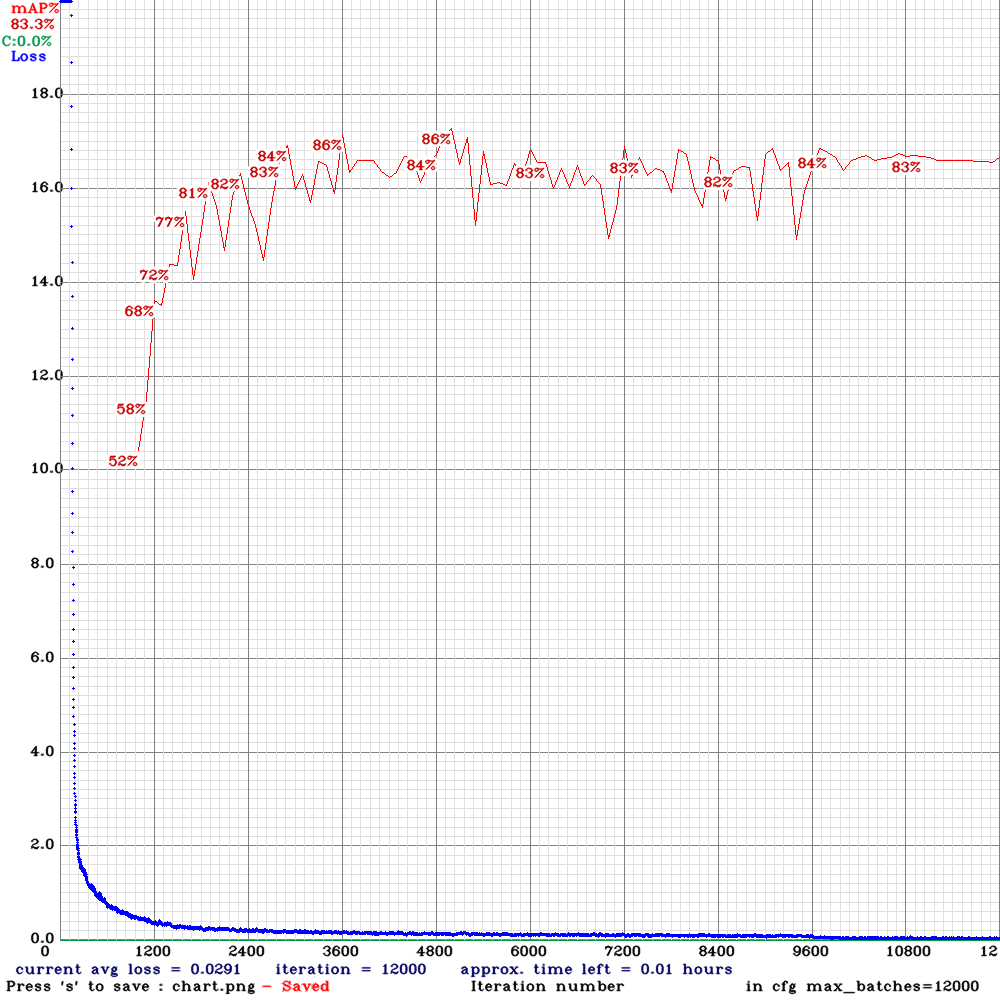
המודל הזה הוא פחות או יותר המודל הקודם, מה שאומר שהניסוי נכשל, אבל אפילו ניסוי שנכשל יכול ללמד לא מעט. נראה לי שהבעייה העיקרית היא ה loss function שלא ירדה מספיק נמוך.

### אימון שישי

המטרה של המודלים הבאים היא לבחון את השפעת ה learning rate. כל הניסויים הבאים יהיו ב batch 64. בשביל לקבוע את הlearning rates לאימונים הבאים אני איצור מספר ראנדומלי (float). ואכפיל אותו כל פעם ב10 בחזקת מינוס אחד עד מינוס שלוש.

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

גרף של אימון:



מהגרף הנוכחי אפשר לראות שהאימון הזה עולה יותר מהר מהאימון הקודם, למרות שהוא מגיע לאותה נקודה של 86 אחוזי זיהוי המודל עושה זאת הרבה יותר מהר.

הlearning rate הנוכחי אינו גבוה בהרבה מהlearning rate הסטנדרטי 0.0026 והוא עוזר למודל לעלות מאוד מהר. ייתכן שלהעלות את ה learning rate עוד ייפגע בתוצאות אלו או רק יהפןך אותן לטובות יותר. את זה אדע אחרי האימונים הבאים

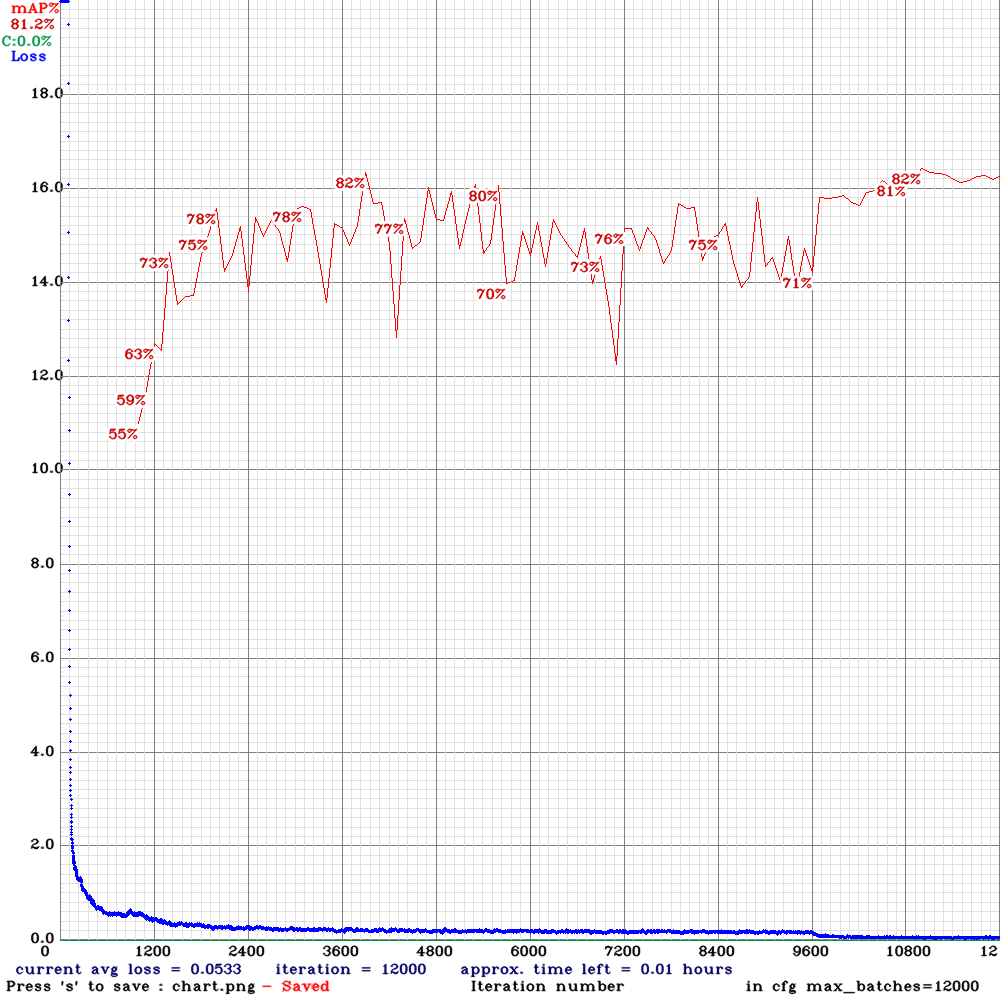
### אימון שביעי

האימון הזה הוא אימון מספר שתיים בבחינת ה learning rate. האימון הזה יהיה עם learning rate של 0.036

נתוני אימון:

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

גרף אימון:



כפי שאפשר לראות learning rate גבוה פוגע באימון ככל הנראה, לפחות כך זה נראה מהמודל הנוכחי, ייתכן שזה בגלל שבכל איטרציה המשקלים קופצים יותר מדי והמודל לא מסוגל ללמוד כמו שצריך

אבל בשביל לקבוע בצורה חד משמעית את השפעת learning rate גבוה צריך עוד ניסוי אחד של learning rate עוד יותר גבוה, זה יהיה הניסוי הבא

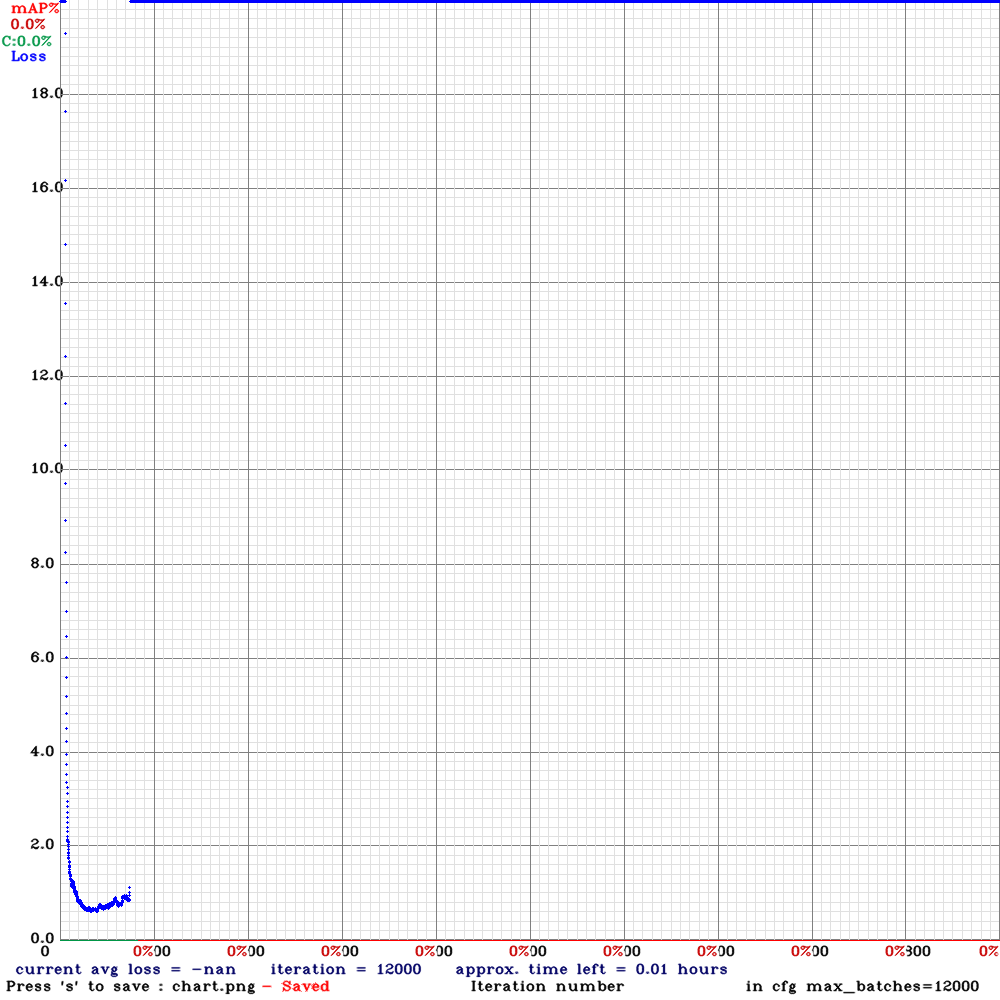
### אימון שמיני

האימון השמיני יהיה האימון עם ה learning rate הגבוה ביותר שאנסה שהוא 0.62 אני חוזה שהמודל יאמן את עצמו בצורה לא טובה כמו המודל הקודם, וייתכן שאפילו יהיה לא יציב.

ההגדרות של האימון:

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

גרף האימון:



אפשר לראות שהאימון הייה מאוד לא יציב והסתיים ב -nan שזה not a number. מה שמעניין זה שהמודל התחיל את האימון היטב, ואז פתאום קפץ ל-nan. הניחוש שלי זה שהיה איפשהו לוג של אפס או חילוק באפס ואז כאשר בוצע ה loss function הכל הפך ל nan.

המסקנה משלושת האימונים האלה היא ש הlearning rate הסטנרטי של yolo הוא יחסית טוב. ובוודאות חובה לשמור על אותו חזקה של עשר בשביל ה learning rate כמו המודל הסטנדרטי.

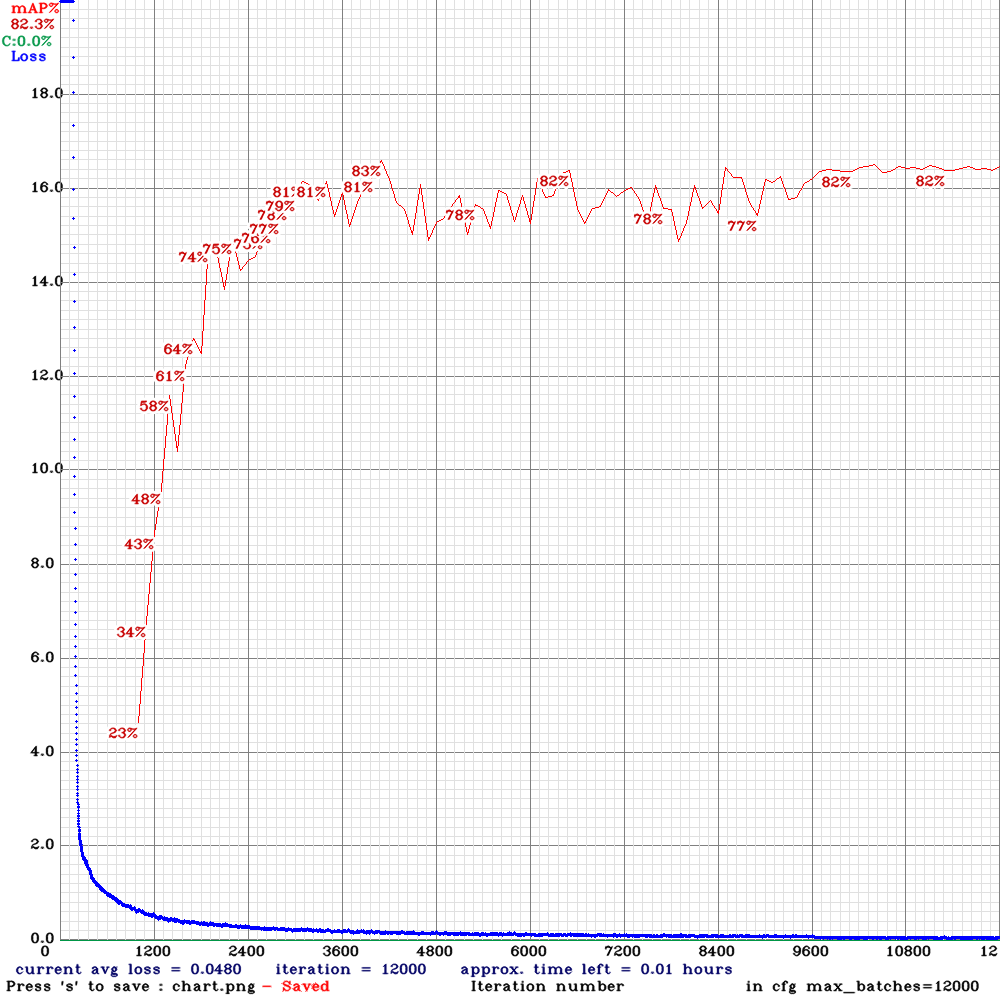
### אימון תשע

האימון הבא יהיה קשור לאנקורים, האנקורים הם פרמטר בyolo שקובע איך yolo מגלה מספר פריטים שנמצאים באותו תא בסריג (הסבר יותר מורכב אפשר למצוא בחלק של המודל).

יש לyolo אנקורים סטנדרטיים שבהם השתמשנו עד עכשיו, לעומת זאת יש אפשרות גם לחשב את האנקורים הכי טובים בשביל המידע שלך באמצעות אלגוריתם שקורים לו k-means (הסבר על האלגוריתם הזה נמצא בחלק המודל). נתוני האימון:

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

גרף אימון:



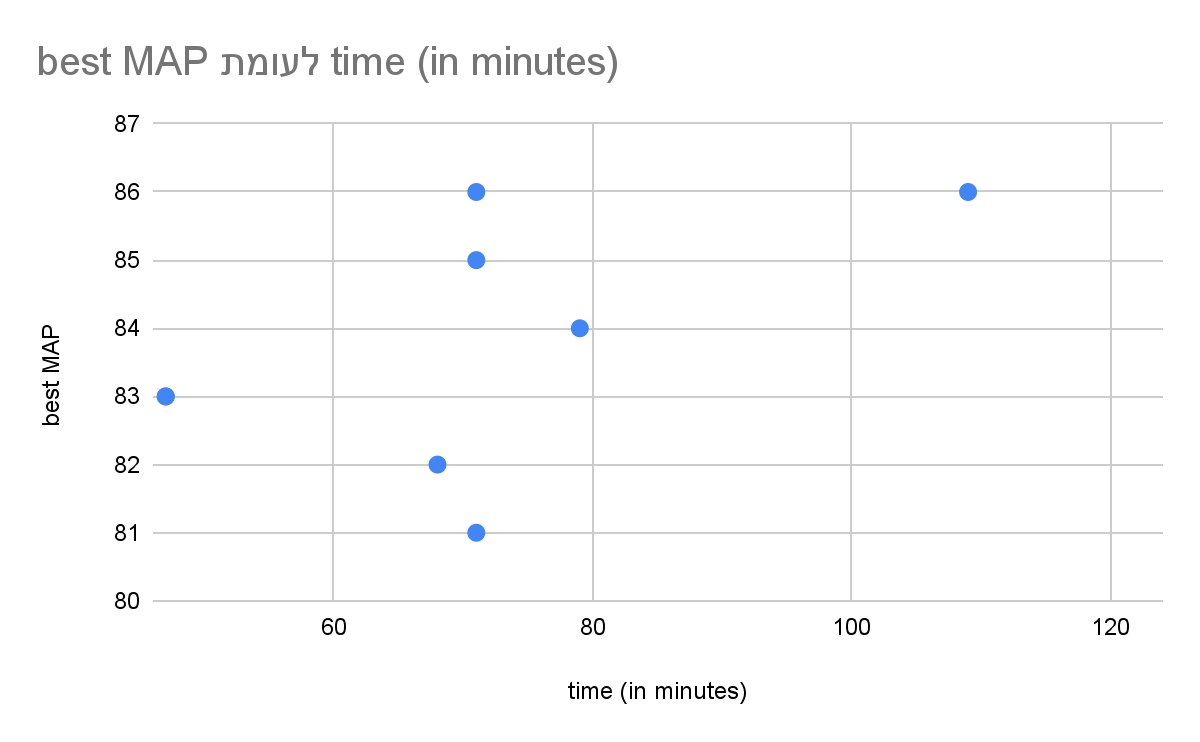
## סיכום אימונים:

באופן מפתיע שינוי ההיפר פרמטרים לא השפיעה הרבה על הדיוק של המכונה (חוץ במקרה של ה nan). לעומת זאת זמן האימון השתנה מאוד בשינוי הפרמטרים ולכן במקרה של אימון מכונה מההתחלה צריך לקחת את השילוב הכי טוב של זמן ודיוק (map).

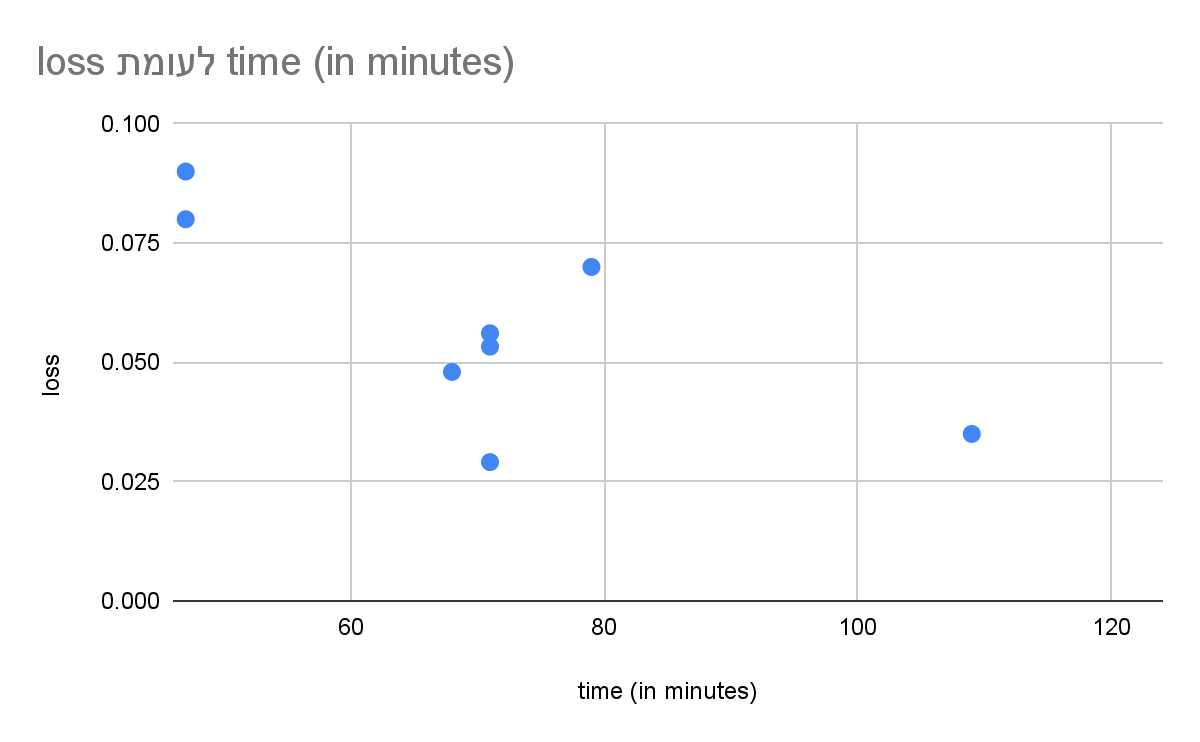
את סיכום האימונים אפשר לראות בטבלה הבאה

| 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 |

ובשני הגרפים הבאים:



ו

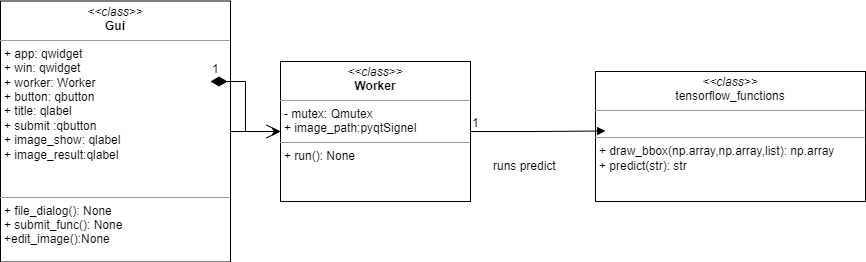


## תיאור יצירת הישום:

התוצר הסופי של עבודת הגמר הוא אפלקציה 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 הפח מיחזור שאליו הוא צריך להיזרק.

בשביל ליצור את ממשק המשתמש השתמשתי בספרייה ששמה pyqt5 שהיא בעצם גרסה של פייתון של תוכנה ששמה qt שבעצם נועדה ליצירת ממשקי משתמש שיכולים לעבוד בכל מערכת הפעלה. הייתרון 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 היא ליצור thread בשביל שהאפלקציה לא תיתקע, ולאחר מכן ה worker מריץ את פונקציית ה predict ב tensorflow\_funtions

בשביל להשתמש במודל הייתי צריך להפוך את מודל ה darknet שלי למודל טאנסורפלו בגלל שהתקנת darknet על וינדוס זה קצת מסובך. בשביל לעשות את זה השתמשתי בספרייה שמצאתי בגיטהאב בשם

tensorflow-yolov4-tflite שעושה בדיוק את זה. לאחר מכן הייתי צריך לכתוב את הקוד שלוקח את המודל שנוצר ומשתמשת בו בשביל ליצור את התיבות התוחמות, ולאחר מכן מציירת את התיבות על התמונה.

בשביל לעשות את זה כתבתי שתי פונקציות: draw\_bbox בשביל הציור ו predict שמפעילה את המודל וקוראת ל draw\_bbox בשביל לצייר את התיבות התוחמות.

predict עובדת כך:

* הפונקציה מקבלת מיקום של תמונה( file path). קולטת את האורך והגובה של התמונה מקטינה אותה ל 256 על 256 ומבצעת נורמליזציה
* הפונקציה מעבירה את התמונה דרך המודל ומקבלת את ה y\_predict.
* הפונקציה מריצה את אלגוריתם נון מקס סופריציה.
* הפונקציה קוראת ל draw\_bbox בשביל שתצייר את התיבות התוחמות על התמונה
* הפונקציה מגדילה את התמונה חזרה לגודל המקורי שלה, שומרת ומחזירה את המיקום החדש

הפונקציה draw\_bbox עובדת כך:

* הפונקציה מקבלת תמונה, תיבות תוחמות, רשימה של תווית ומשתנה בוליאני שאומר האם להראות את התוויות בתמונה
* הפונקציה יוצרת רשימה של צבעים שאיתם היא עובדת
* הפונקציה מבטלת את הנורמליזציה של התיבות התוחמות באמצעות הכפלה בגודל התמונה
* הפונקציה עוברת על התיבות התוחמות ומציירת כל אחת בצבע
* אם צריך לצייר את התווית בתמונה הפונקציה שמה את הטקסט של כל טקסט מעל התיבה
* הפונקציה מחזירה את התמונה

## מדריך למפתח

*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*

*#gui.py*

*#functions and classes for the gui*

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

*self.image\_show.adjustSize()*

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

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

*def process\_func(self):*

*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.setPixmap(pix) #sets the image to the label*

*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:

בשביל להתקין את ה gui צריך לעשות את הדברים הבאים\*:

* להוריד את תוכנת git
* להריץ את הפקודה הבאה:

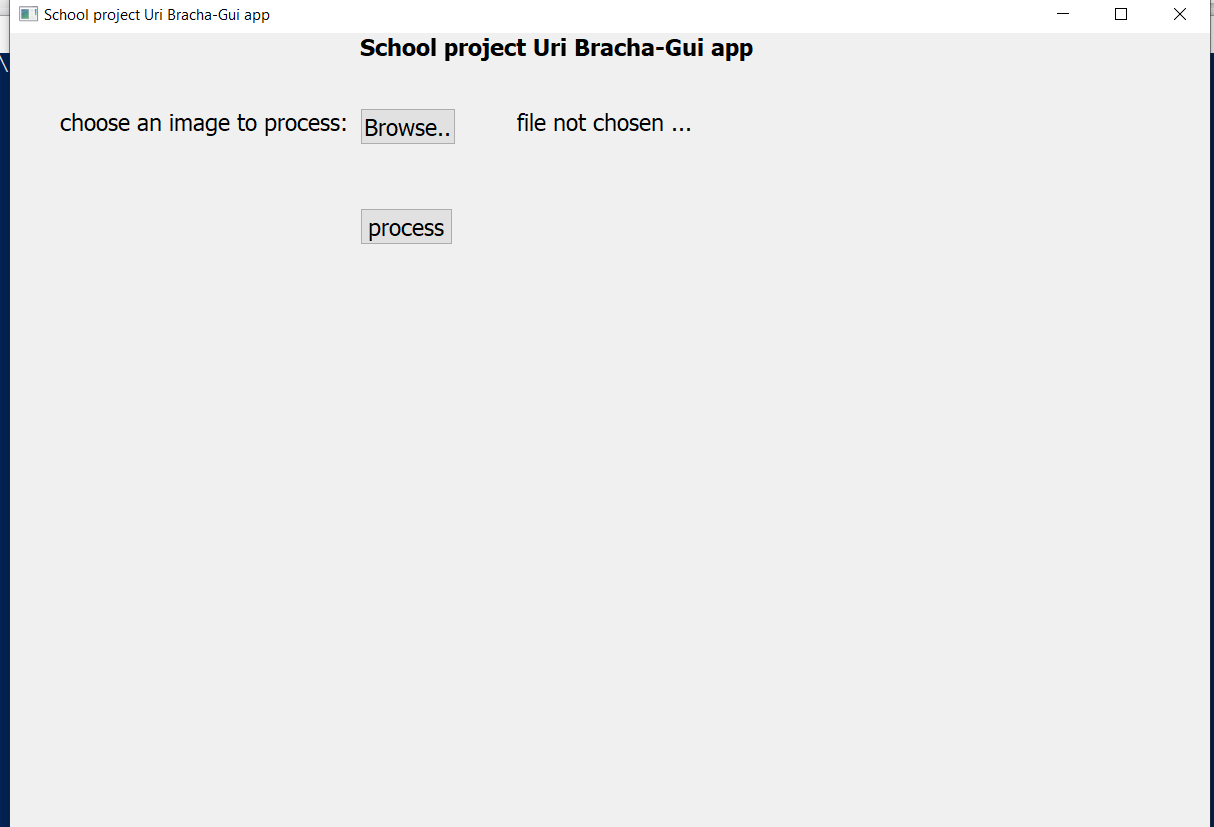
git clone https://github.com/uribracha2611/yolov4\_tiny\_waste\_detection.git

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

pip install -r requirements.txt

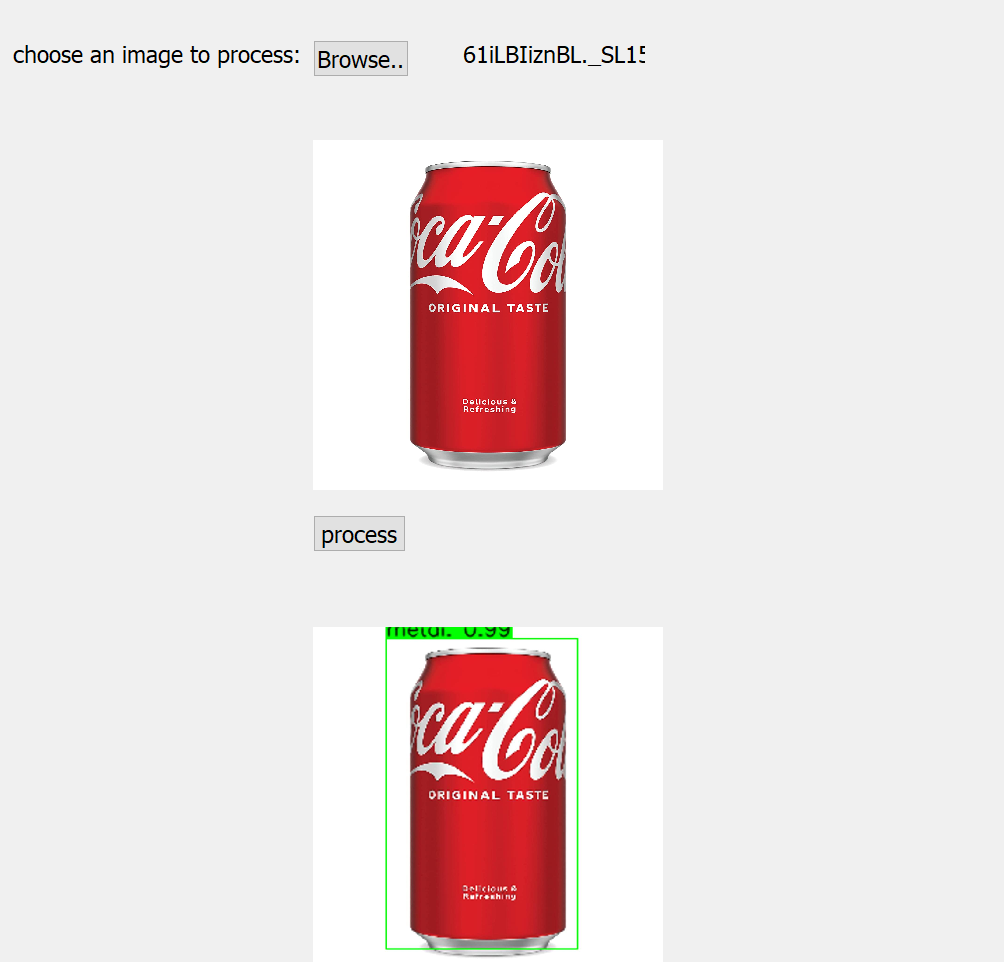
* להריץ בתוך אותה קנסולה python gui.py

\*הערה כל שימוש אחרי הראשון דורש רק את השורה האחרונה

מדריך שימוש בתכונה:  
התוכנה נראית כך:  


כדי להכניס תמונה צריכים ללחוץ על כפתור ה browse לאחר מכן יפתח חלון שבו אפשר לבחור קובץ תמונה, בו צריך לבחור קובץ וללחוץ על ok. בנקודה הזאת רואים את התמונה מתחת לקובץ ה browse ואת שם הקובץ של התמונה. לאחר מכן כל מה שצריך לעשות זה ללחןץ על process לחכות זמן מה (חצי דקה עד מספר דקות) ואז מתחת ל process תופיע התמונה עם התיבות התוחמות. אם רוצים לקבל את התמונה כקובץ צריך ללכת לתקיית saves שם תופיע התמונה .

תמונה של ה gui אחרי זיהוי:



# רפלקציה

הפרויקט הזה עלה בהתחלה כרעיון כאשר עשיתי את פרוייקט עבודת הקיץ שלי, שהיה על אותו נושא של זיהוי פסולת הפרויקט הזה הוא שדרוג של הפרויקט המקורי, שבו יכולתי לזהות רק פריט אחד, ואחת השאלות שעלו לי באותו התקופה על האם אפשר לזהות מספר פריטים ולאחר שלמדתי על yolo חשבתי לעשות את הפרויקט הזה.

העבודה על הפרוייקט הייתה מעניינת ומסובכת, בהתחלה התקשיתי לא מעט בעריכת מסד הנתונים TACO ובמציאת הדרך הטובה ביותר להשתמש בyolo. לפי דעתי רוב הבעיות שלי בפרוייקט היו בגלל ש TACO לא התאים לעבודה שרציתי לעשות, רוב התמונת היו ברובן רקע עם ממש מעט מקום לפריטי הפסולת, בעיקר אחרי שמקטינים את התמונות ל 256 על 256

הקשיים שלי עם yolo היו בעיקר בלהבין את מבנה הרשת, כאשר לא הבנתי את הרשת בצורה נאותה, לא יכולתי להשתמש בה כמו שצריך ולא לפתור את הבעיות שלי, אבל אחרי שהבנתי איך הרשת עובדת, הבנתי שהבעיות שלי הם בעצם בקובץ המידע שלי.

לעומת זאת, העבודה הייתה מתגמלת מאוד ולמדתי ממנה המון. למדתי על סוגי רשתות חדשות כמו yolo והגרסאות השונות שלה. למדתי איך יוצרים ממשק משתמש בפייתון, דבר שאני בטוח שאשתמש בעתיד אפילו ללא קשר ללמידת מכונה. ולמדתי איך עושים סדר בתוך ה google drive בעיקר במקרה של אימונים בyolo.

אם הייתי מתחיל מחדש את העבודה היום הייתי עושה מספר דברים אחרת. קודם כול הייתי מחפש מסד נתונים יותר מתאים או יוצר אחד בעצמי מלכתחילה. בנוסף, עם הידע שיש לי עכשיו על yolo ככל הנראה הייתי יכול לעבוד עם טאנסורפלו בכל הדרך, מה שהיה מקל עלי בנושא ממשק המשתמש. לעומת זאת ייתכן שכן הייתי עובד עם darknet, כי האימון ויצרת הגרפים שם הייתה קלה מאוד.

המסקנה העיקרית שלי מהפרויקט היא שהנושא של למידת מכונה מאוד מעניין אותי. דרך החשיבה שונה בהרבה מדרך החשיבה של תכנות "סדנטרטי" ואני יכול לראות את עצמי עובד בנושא בעתיד, אפילו בזמני החופשי.

דברים שהייתי מוסיף לעתיד:

* יכולת לעבוד עם קבצי וידיאו
* יכולת לקבל וידיאו ממצלמה (לדוגמה מצלמת מחשב).
* הגדלת מסד הנתונים בשביל קבלת סיכויי זיהוי טובים יותר, אולי בשימוש 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