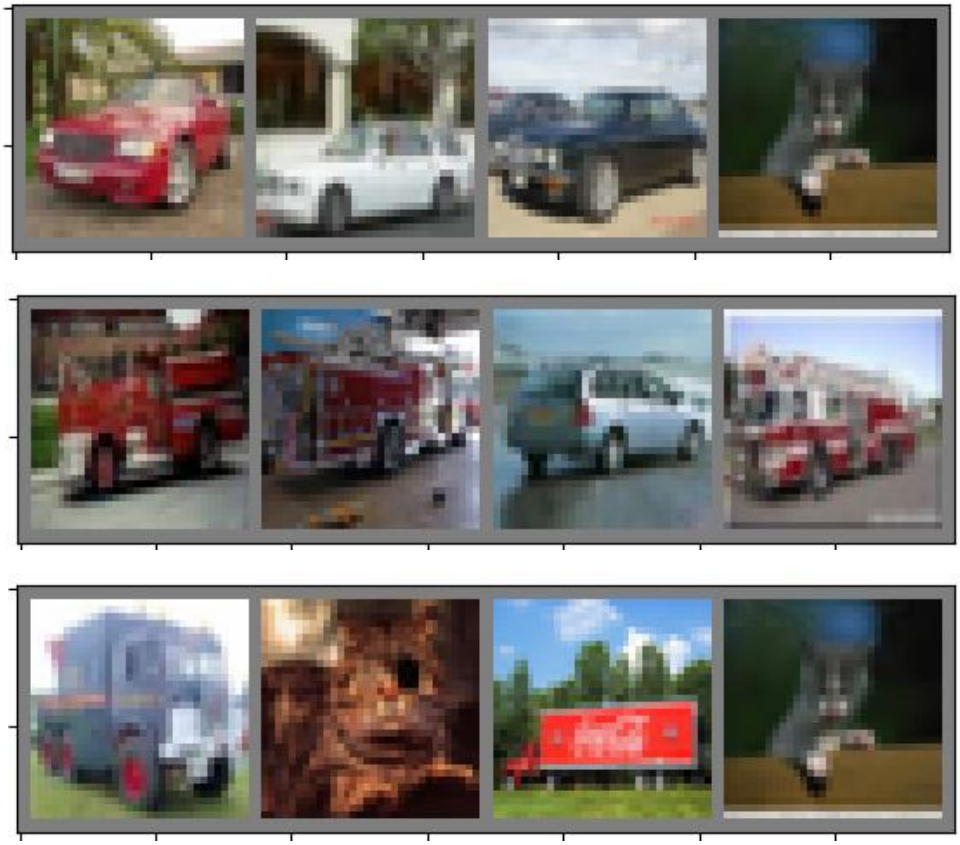
APML Ex1

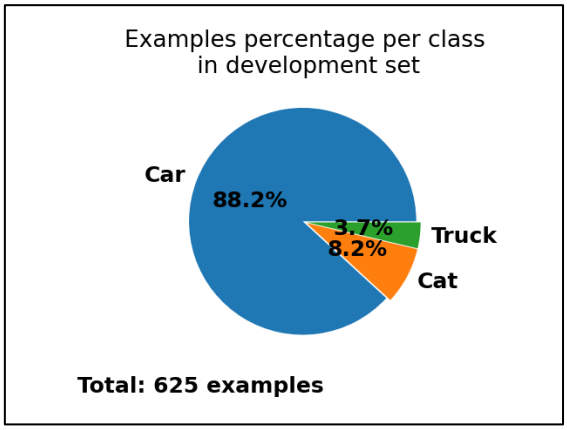
Oded Mousai (312246291)

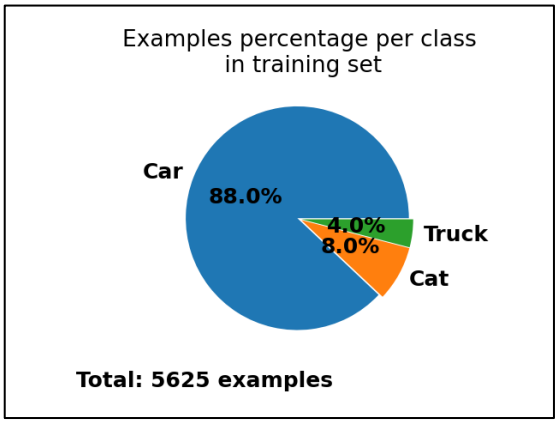
A basic data analysis

דוגמאות לתמונות מהדאטה:



Training Set ו- Test set מפולחים לפי מחלקות:





ניתן לראות שהדאטה הוא imbalanced – הדאטה מכיל מספר רב של דוגמאות מסוג "Car" לעומת מעט מאוד דוגמאות משתי המחלקות האחרות. בסעיף הבא נראה כיצד דבר זה משפיע על התוצאות של ה-baseline model.

Evaluation of the baseline model

בשאלה זו נעריך את ה-baseline model שאומן על ה-training set שסופק לנו. אז טענתי את המודל המאומן וקיבלתי שה-Accuracy שלו על ה-test set הוא:

```
Accuracy of the network on the 625 test images: 88.16%
```

ניתן היה לחשוב שזו תוצאה טובה, אך מכיוון שה-training set הוא imbalaned לטובת המחלקה "car" נצפה שהמודל הצליח לסווג בצורה טובה רק דוגמאות ממחלקה זו. ואכן אם נחשב את הדיוק לפי כל מחלקה נקבל:

```
Accuracy of   car :   100 %
Accuracy of truck :     0 %
Accuracy of   cat :     0 %
```

ניתן לראות שהתוצאות לא טובות, כי אומנם המודל הצליח לסווג בצורה מעולה דוגמאות מהמחלקה "Car", אך הוא לא הצליח בכלל לסווג נכונה דוגמאות משתי המחלקות האחרות.

נשים לב שניתן היה לחשוד כבר מהתוצאה שקיבלנו עבור הדיוק הכללי – קיבלנו 88.16% שהוא קרוב לאחוז הדוגמאות שהן "Car" ב-development set (88.2%).

הסבר אפשרי לתוצאה זו היא שבגלל שה-training data הוא imbalanced , המודל מעדיף לסווג את כל הדוגמאות עם ה-majority class שהיא המחלקה "Car" מפני שתוצאה זו נותנת דיוק גבוה.

כדי לוודא שאנחנו מעריכים נכון את המודל נשתמש במדדים אחרים:

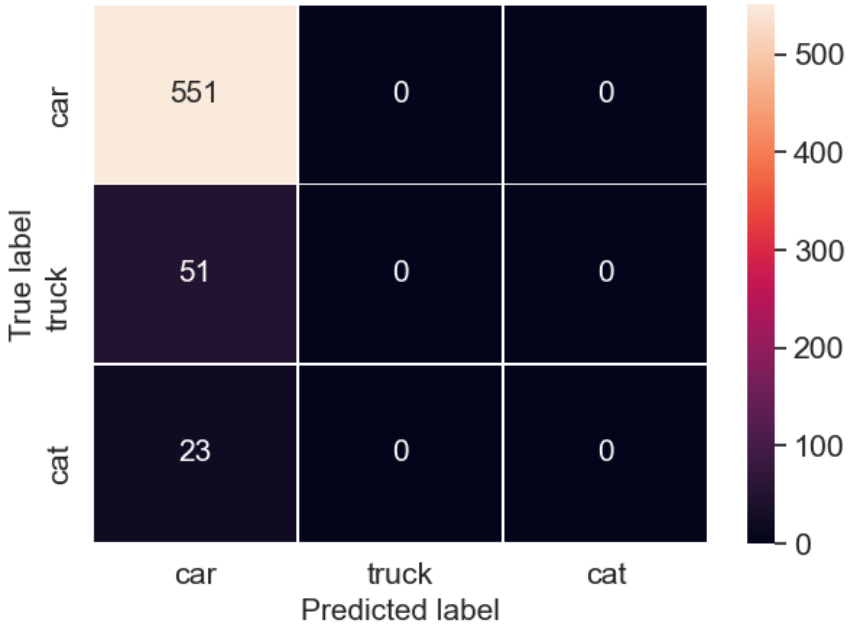
Performance metrices

Precision for class A – מתוך מה שהמודל סיווג כמחלקה A, כמה המודל צדק.
Recall for class A – מתוך כל הדוגמאות שהן A, כמה המודל צדק.
F1 for class A – שקלול של P ו-R.
כדי לחשב את המדדים הללו נעזרים ב- confusion matrix.

פירוש תוצאות:

R גבוה + P גבוה: המודל מצליח להתמודד עם המחלקה.
R נמוך + P גבוה: המודל לרוב לא מצליח לזהות את המחלקה, אבל כשהוא כן מצליח אז הסיווג שלו אמין מאוד.
R גבוה + P נמוך: המודל לרוב מצליח לזהות את המחלקה, אבל הסיווג שלו לא אמין (הוא מסווג אותה גם כמחלקות אחרות).
R נמוך + P נמוך: המודל לא מצליח להתמודד עם המחלקה.

עבור ה-baseline model קיבלנו את התוצאות הבאות:



	precision	recall	f1-score	support
Car	0.882	1.000	0.937	551
Truck	0.000	0.000	0.000	51
Cat	0.000	0.000	0.000	23
accuracy			0.882	625
macro avg	0.294	0.333	0.312	625
weighted avg	0.777	0.882	0.826	625

ניתן לראות שה-baseline model חזה את כל הדוגמאות כ-"Car" כפי שציפינו שיקרה עקב בעיית ה-imbalanced data.

Train the given model the best I can

Augmentation על ה-training set

ראשית ביצעתי אוגמנטציה על ה-training set.

נשים לב שאוגמנטציה עדיין לא תשפר את הדיוקים של ה-baseline model, מכיוון שעדיין הדאטה יהיה מוטה לטובת המחלקה "car" כי הגדלנו את כל המחלקות באותו הפרופורציה (ואכן שהרצתי אוגמנטציה בלי משקלים קיבלתי דיוק של 100 אחוז על car, אך לשאר המחלקות דיוק של 0, כמו ב-baseline). אם כן, הסיבות לשימוש באוגמנטציה הן:

1) להגדיל את העושר של הדוגמאות ובכך לקוות שהמודל יצליח לבצע הכללה בצורה טובה (יתמודד עם unseen data בצורה טובה)
2) הגדלת הדאטה מאפשרת לעשות cross-valdiation בצורה יותר טובה, כי עבור מעט דוגמאות נקבל folders קטנים שלא מספיקים.

יש 2 דרכים לבצע אוגמנטציה:

Offine augmentation – הגדלת ה-training set ע"י ביצוע אוגמנטציות לתמונות (התמונות המקוריות נשארות ומוסיפים למאגר הדוגמאות את האוגמנטציות עליהן).
Online augmentation – תוך כדי האימון לכל תמונה מגרילים האם להפעיל עליה אוגמנטציה כלשהי או לא. לכן ה-training set לא גודל.
החלטתי לבצע את השיטה הראשונה משום שגם ככה המחלקות "cat" ו-"truck" לא מיוצגות מספיק ב-training set.

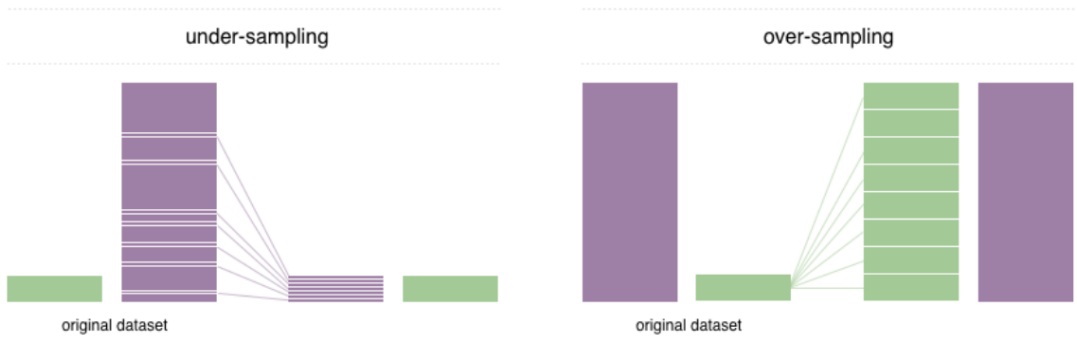
סוגי אוגמנטיציה שלא השתמשתי בהן: Flip vertically משום שלרוב חתולים, משאיות ומכוניות מופיעים כאשר הגלגלים/הרגליים למטה.

הערה: בפועל האוגמנטציות רק הקטינו את הדיוק של המודל, ולכן בסוף לא השתמשתי באוגמנטציות כלל.

התמודדות עם Imbalanced data

שיטות:

1) Over-sampling the minority classes
2) Over sampling the majority classes
3) Classes reweight – לתת משקל שונה לכל מחלקה כך שה-minority classes יקבלו משקל יותר גבוה, ובכך "לאזן" את המחלקות שבמאגר הדוגמאות. 2 דרכים אפשריות שהן שקולות:
   - ע"י נתינת משקל שונה לכל דוגמה ב-training set לפי המחלקה של הדוגמה.
   - ע"י נתינת משקל שונה לכל מחלקה ב-training loss function .
4) לייצר דאטה סינתטי – בעזרת SMOTE.



Experiments by Trail & Error

בשלב זה אימנתי את המודל בשיטות השונות שתוארו לעיל, ולאחר כל אימון ביצעתי אבולאציה של המודל שהתקבל כך שאוכל להשוות בין השיטות השונות. כל שיטה אומנה עם ההיפר-פרמטרים הבאים:

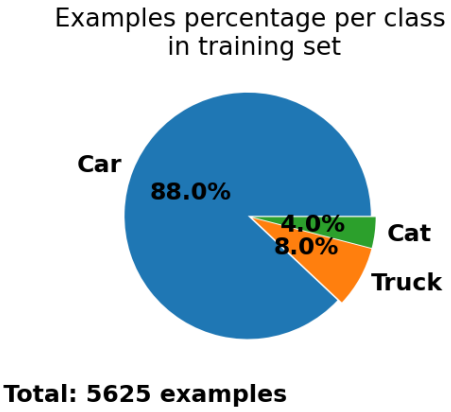
batch_size = 4, shuffle=true
Epochs_num = 10
Optimizer=SGD, Learning_rate=0.001, Momentum=0.9

לקחתי בכוונה מספר Epochs יחסית נמוך, כי לא היה לי זמן (וכוח חישוב) לאמן את כל השיטות בזמן סביר על הרבה איטרציות.

Original train set

בניסוי זה לקחנו את הדאטה המקורי בלי שינויים.

ניתן לראות שהגדלת ה-epochs ב-training set שיפרה את המדדים של המחלקה "Truck", אבל לא של המחלקה "Cat" (אני מניח שהמודל הבסיסי אומן רק על epoch בודד, וכאן אימנו על 10 epochs).

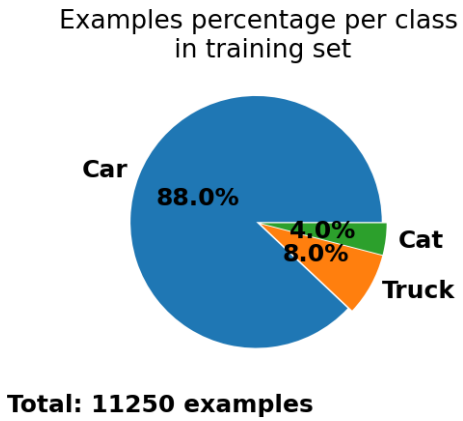



```
Accuracy of the network on the 625 test images: 89.92%
Accuracy of   car :    99 %
Accuracy of truck :    25 %
Accuracy of   cat :     0 %
C:\Users\odedi\Anaconda3\lib\site-packages\sklearn\metrics\
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

         Car      0.901     0.996     0.947       551
       Truck      0.812     0.255     0.388        51
         Cat      0.000     0.000     0.000        23

    accuracy                          0.899       625
   macro avg      0.571     0.417     0.445       625
weighted avg      0.861     0.899     0.866       625
```

Train set with augmentation

בניסוי זה הוספנו ל-training set דוגמאות מה-training set שעברו אוגמנטציה.

ניתן לראות שהאוגמנטציה עדיין לא עוזרת להתמודד עם ה-imbalanced data. משום שעדיין לא מקבלים תוצאות גרועות עבור המחלקה "Cat".

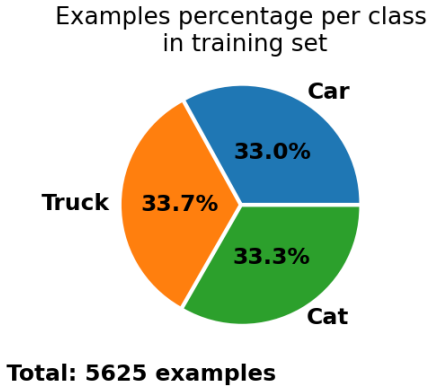



```
Accuracy of the network on the 625 test images: 90.72%
Accuracy of   car :   100 %
Accuracy of truck :    31 %
Accuracy of   cat :     0 %
C:\Users\odedi\Anaconda3\lib\site-packages\sklearn\metrics\
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

         Car      0.906     1.000     0.951       551
       Truck      0.941     0.314     0.471        51
         Cat      0.000     0.000     0.000        23

    accuracy                          0.907       625
   macro avg      0.616     0.438     0.474       625
weighted avg      0.876     0.907     0.877       625
```

Train set with sampler

בניסוי זה ביצענו over sampling + down sampling כך שהדאטה באימון היה מאוזן לגמרי.

נשים לב ששיטה זו שיפרה מאוד את ה-accuracy של המחלקות "cat" ו-"truck", אך ניתן לראות שה-recall של שתי המחלקות הללו גבוה לעומת ה-precision שלהן שנמוך, מה שמעיד על כך שהמודל בכוח משייך דוגמאות למחלקות הללו. זה דבר לא רצוי.

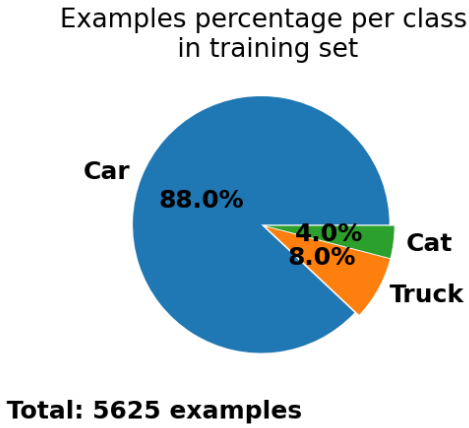



```
Accuracy of the network on the 625 test images: 75.84%
Accuracy of   car :    75 %
Accuracy of truck :    74 %
Accuracy of   cat :    82 %
              precision    recall  f1-score   support

         Car      0.968     0.757     0.849       551
       Truck      0.304     0.745     0.432        51
         Cat      0.275     0.826     0.413        23

    accuracy                          0.758       625
   macro avg      0.516     0.776     0.565       625
weighted avg      0.888     0.758     0.799       625
```

Train with weights in training loss function

בניסוי זה במקום לאזן את הדאטה של האימון באופן ישיר, נתנו משקל לכל מחלקה ב- training loss function, כך שככל שמחלקה יותר נדירה כך המשקל שלה יותר גבוה ועל כן המודל יתאמץ לחזות נכון את המחלקה הזו. הנוסחא למשקל של מחלקה c הוא: |c| / |training set| (מספר דוגמאות כולל ב-training set חלקי מספר דוגמאות שמתויגות עם המחלקה c).

ניתן לראות שעבור המחלקה "Truck" יש לנו P גבוה ו-R נמוך, ועבור המחלקה "Cat" ההפך. התוצאות הללו לא טובות.

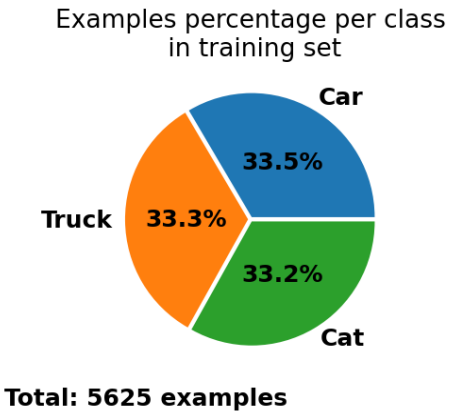



```
Accuracy of the network on the 625 test images: 87.68%
Accuracy of   car :    93 %
Accuracy of truck :    37 %
Accuracy of   cat :    60 %
              precision    recall  f1-score   support

         Car      0.931     0.935     0.933       551
       Truck      0.826     0.373     0.514        51
         Cat      0.286     0.609     0.389        23

    accuracy                          0.877       625
   macro avg      0.681     0.639     0.612       625
weighted avg      0.899     0.877     0.879       625
```

Train set with augmentation + sampler

בשיטה זו ביצענו אוגמנטציה + איזון הדאטה ע"י over&down sampling.

ניתן לראות שהתוצאות עדיין לא טובות.

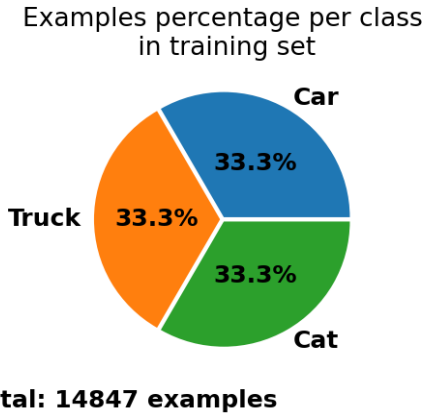



```
Accuracy of the network on the 625 test images: 57.44%
Accuracy of   car :    56 %
Accuracy of truck :    47 %
Accuracy of   cat :    91 %
              precision    recall  f1-score   support

         Car      0.972     0.570     0.719       551
       Truck      0.235     0.471     0.314        51
         Cat      0.105     0.913     0.188        23

    accuracy                          0.574       625
   macro avg      0.437     0.651     0.407       625
weighted avg      0.880     0.574     0.666       625
```

Train set with SMOTE

בניסוי זה השתמשתי בשיטה שנקראת Synthetic Minority Oversampling Technique. בשיטה זו מייצרים באופן מתוחכם דאטה סינתטי עבור ה-minority classes, עד שמגיעים לכך שמספר הדוגמאות של ה-minority classes שווה למספר הדוגמאות השייכות ל-majority class.

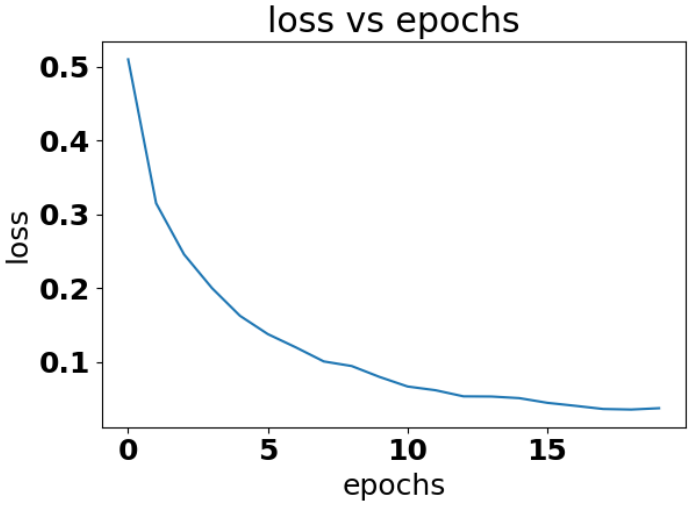
ניתן לראות שהמדדים של המחלקה "Truck" השתפרו.




```
Accuracy of the network on the 625 test images: 87.84%
Accuracy of   car :    93 %
Accuracy of truck :    52 %
Accuracy of   cat :    30 %
              precision    recall  f1-score   support

         Car      0.933     0.935     0.934       551
       Truck      0.614     0.529     0.568        51
         Cat      0.241     0.304     0.269        23

    accuracy                          0.878       625
   macro avg      0.596     0.589     0.590       625
weighted avg      0.881     0.878     0.880       625
```

סיום

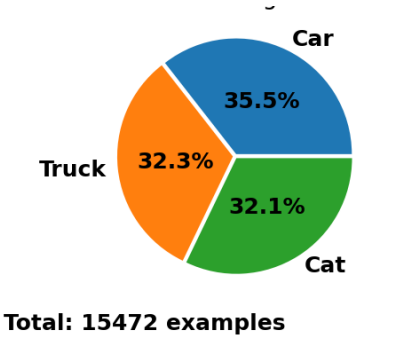
החלטתי לקחת כמודל הסופי את המודל שמשתמש ב-SMOTE, מפני שהוא נתן תוצאות טובות יחסית לשיטות האחרות. בנוסף לא השתמשתי באוגמנטציה כי נראה שזה גורע מהמודל (אולי מכיוון שבאוגמנטציות מעבירים את התמונה לפורמט PIL ?). אז אימנתי את המודל הזה עם 20 epocs וקיבלתי את התוצאות הבאות:




```
Accuracy of the network on the 625 test images: 88.16%
Accuracy of   car :    93 %
Accuracy of truck :    56 %
Accuracy of   cat :    39 %
              precision    recall  f1-score   support

         Car      0.938     0.931     0.934       551
       Truck      0.580     0.569     0.574        51
         Cat      0.321     0.391     0.353        23

    accuracy                          0.882       625
   macro avg      0.613     0.630     0.621       625
weighted avg      0.886     0.882     0.884       625
```

אני מניח שהיה אפשר לאמן את המודל עם מספר איטרציות יותר גבוה מ- 20 (עד שרואים שה-loss מתחיל לעלות) אבל לא עשיתי זאת מפאת קוצר בזמן.

לבסוף, אימנתי מחדש את המודל הנ"ל על כל הדאטה (test set + syntactic data + train data!) ואותו הגשתי כמודל הסופי, וזאת מתוך מחשבה שיותר דאטה יגביר את ההצלחה של המודל.



התפלגות המחלקות עבור סט האימון הסופי

מחשבות נוספות שלא ביצעתי

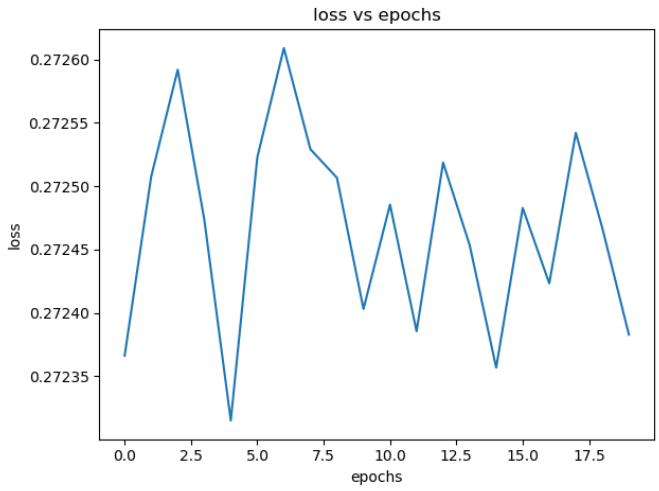
- ביצוע fine-tuning על ההיפר פרמטרים כגון סוג האופטימייזר.
- בדיקה האם במאגר הדוגמאות יש דוגמאות שלא מתויגות נכון, ובמידה שכן – לתקן אותן.
- שילוב של כמה שיטות.

Playing with Learning Rate

בחלק זה ננתח את ההתנהגות של המודל בזמן האימון כאשר שמים learning rate גדול/קטן מדי.

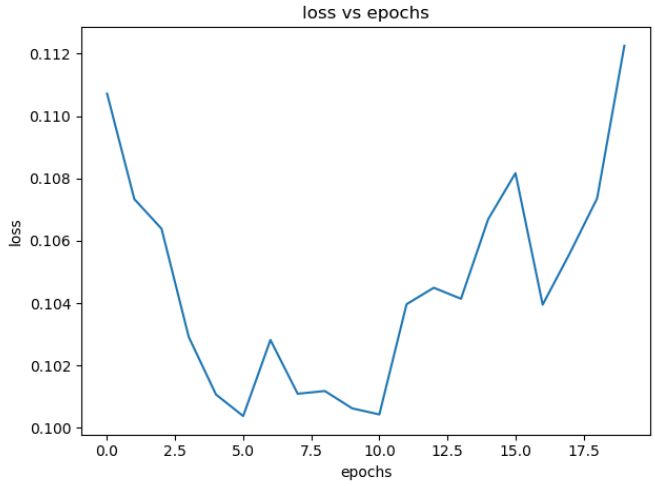
Learning rate = 10^-10 = 0.0000000001

עבור הערך הזה ו-20 איטרציות קיבלנו את התרשים הבא.
ניתן לראות שה-loss לא מתכנס ל-0. הסבר לכך הוא שעבור learning rate קטן מדי בכל שלב באלגוריתם Gradient Decent המשקולות של המודל יתעדכנו באופן מזערי, ולכן למודל ייקח יותר זמן ללמוד את הפונקציה.



Learning rate = 0. 1

עבור הערך הזה ו-20 איטרציות קיבלנו את התרשים הבא.
ניתן לראות שה-loss מתבדר.הסבר לכך הוא שעבור learning rate גדול מדי בכל שלב באלגוריתם Gradient Decent ניקח צעד גדול בכיוון הירידה של הפונקציה, אך צעד גדול מדי עלול להוביל אותנו לנחות מעבר לנקודת המינימום כך שבעצם ה-loss יגדל.



Adversarial Example

בסעיף זה נרצה לבנות תמונה מהונדסת כך שהרשת המאומנת תסווג עבור התמונה הזו תג שגוי. החלטתי לבנות דוגמה שהתג האמיתי שלה הוא "Car" , אבל הרשת המאומנת שלי תסווג את התג במחלקה אחרת. ניסיתי לבנות דוגמה כזאת בעזרת שתי שיטות שונות שבונות את התמונה המהונדסת ע"י הוספת רעש לתמונה המקורית:

Adversarial image = original image + noise

שיטה ראשונה: Optimizing Noise

בשיטה זו התמונה המהונדסת מורכבת ע"י התמונה המקורית שלה מוסיפים תמונת רעש. רוצים למצוא מטריצת רעש כך שמצד אחד הרשת תסווג את התמונה המהונדסת בתג אחר, ומצד שני נרצה שהרעש יהיה ככל האפשר בלתי מובחן לעין האנושית. לשם כך אנו בונים פונקציית Cost מתאימה וממזערים אותה ע"י gradient decent כאשר גוזרים לפי מטריצת הרעש (הפרמטרים של המודל המקורי הם קבועים כעת!). הנוסחא לפונקציית ההפסד:

$$\mathbf{CrossEntropyLoss}[\mathbf{model}(\mathbf{image}+\mathbf{noise}),\mathbf{target\_class}]+\mathbf{C}*\mathbf{norm}(\mathbf{noise})\}$$

image – התמונה המקורית

noise – מטריצת הרעש

target_class – המחלקה השגויה שאנו רוצים שהמודל המאומן יחזה עבור התמונה המהונדסת

model – המודל המאומן

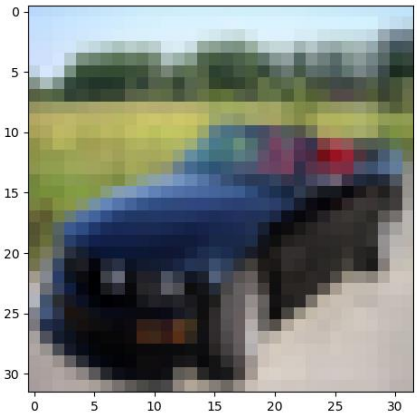
C – קבוע רגוליזציה

במילים: הביטוי הראשון בפונקציית ההפסד מבטא את הרצון שלנו שהמודל המאומן יחזה עבור התמונה המהונדסת מחלקה שגויה שאנו בוחרים מראש. הביטוי השני מבטא את הרצון שלנו שהרעש יהיה מזערי ככל שאפשר, כך שהוספה שלו לתמונה המקורית לא יהיה מובחן לעין האנושית.

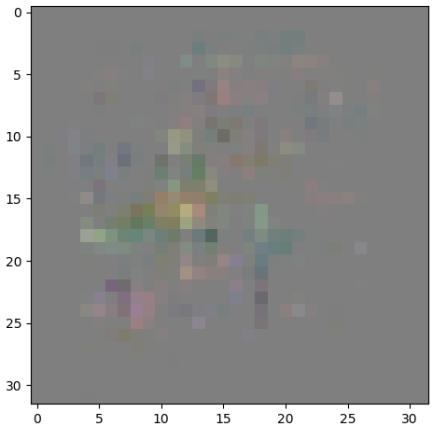
הערות:

- נשים לב בשיטה זו אפשר לכוון ל-target label מסויים (בשיטה השנייה זה לא יהיה אפשרי).
- אפשר לאתחל את מטריצת עם רעש לבן/אפור/גאוסיאני. אני אתחלתי את המטריצה עם ערכים אפורים (מטריצת אפסים) כי זה נתן לי את התוצאה הטובה ביותר מבחינת הנראות בעין של התמונה המהונדסת. לעומת זאת כשאתחלתי עם רעש גיאוסיאני, ההטעייה של המודל המאומן לקחה יותר מהר אבל מצד שני הנראות של התמונה המהונדסת הייתה לא טובה (מלאה ברעש צבעוני). יש כאן trade off בין הטעיית העין האנושית להטעיית המודל, שאפשר לשלוט עליו גם ע"י הקבוע C.
- בפונקציית ההפסד בחרתי להשתמש בנורמה l1 . לכן אנו בעצם דורשים שערכי מטריצת הרעש יהיו קרובים ככל הניתן ל- 0 (נורמה זו היא סכום הערכים המוחלטים של כל הפיקסלים). בייצוג הנוכחי 0 מייצג ערך אפור, ולכן לא נופתע אם תמונת רעש אופטימלית תהיה אפורה ככל הניתן.

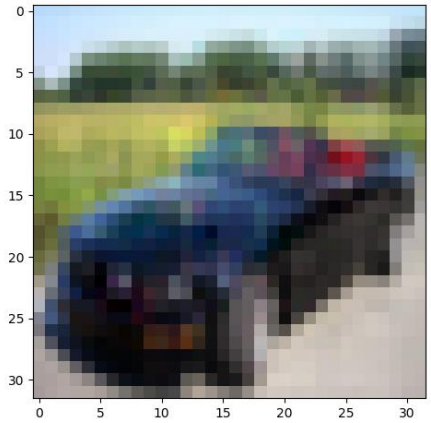
תוצאות:



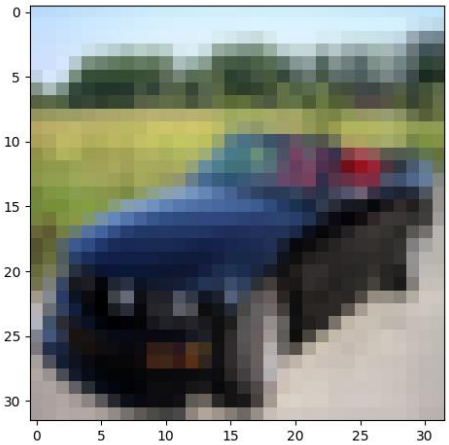
Original image
*Tag: "**Car**"*



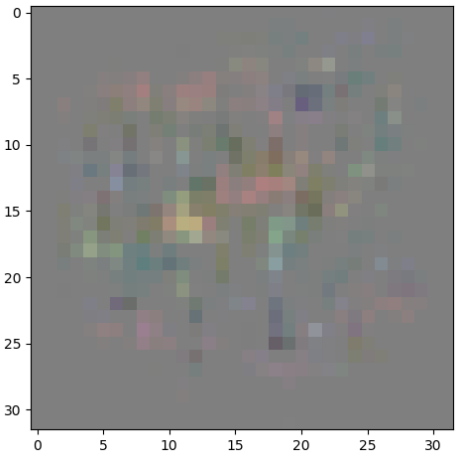
Noise



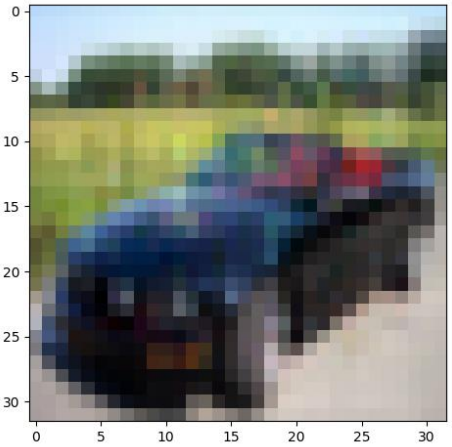
Adversarial image
*Predicted tag: "**Truck**"*



Original image
*Tag: "**Car**"*



Noise



Adversarial image
*Predicted tag: "**Cat**"*

מעניין לראות שבדוגמה השנייה התבנית שמופיעה תמונת הרעש מזכירה צורה של הולך על ארבע.

שיטה שנייה: Fast Gradient Sign Method (FGSM)

כמו בשיטה הראשונה, גם בשיטה זו בונים תמונה מהונדסת ע"י מציאת רעש והוספתו לתמונה המקורית, אך הרעיון שונה: כאן נרצה למצוא רעש שהוספתו לתמונה המקורית תיצור תמונה מהונדסת שממקסמת את ה-loss function (לפי התמונה המקורית הפעם!) שבעזרתה אימנו את המודל. לאחר שיצרנו את התמונה המהונדסת נקווה שהמודל המאומן יחזה עבורה מחלקה שגויה.

כיצד עושים זאת: נרשום את ה-loss function ונגזור אותה לפי התמונה המקורית X (נשים לב שלא גוזרים לפי הפרמטרים של המודל, כי כעת הם קבועים!). הגרדיאנט שמתקבל מסמן את כיוון העלייה של פונקציית ההפסד (הכיוון שנרצה לשנות את X כדי למקסם את פונקציית ההפסד לפי X), ולכן כדי להתרחק מהמינימום – נצעד בכיוון הגראדיאנט (בדיוק ההפך ממה שעושים ב-GD). נשים לב שכדי לא להרעיש את התמונה המקורית יותר מדי אנו לוקחים רק את הסימן של הגרדיאנט (ולא את הנורמה שלו), ומכפילים באפסילון (מספר קטן מאוד) כדי לקבל רעש לא מורגש.

$$\boldsymbol{adversarial\ img} = \boldsymbol{x} + \boldsymbol{\varepsilon} * \boldsymbol{sign}[\boldsymbol{\nabla_x J}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})]$$

X – התמונה המקורית

Y – התג האמיתי של התמונה המקורית

J – פונקציית ההפסד

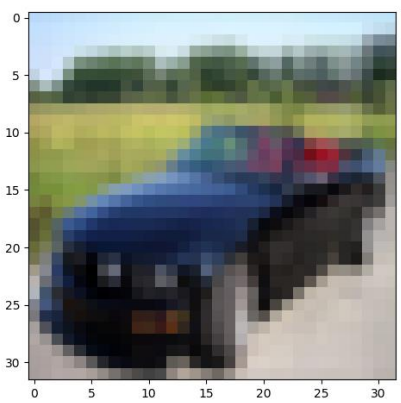
θ - הפרמטרים של המודל המאומן

נשים לב ככל שנגדיל את אפסילון נעלה את הסיכוי שהמודל המאומן יטעה עבור הדוגמה המהונדסת, אך מצד שני התמונה המהונדסת תראה רועשת יותר.

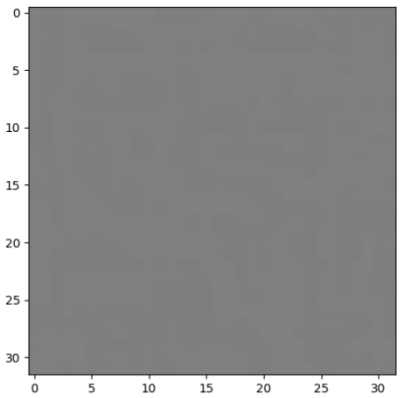
יתרונות: שיטה מהירה מאוד כי צריך לחשב גרדיאנטים רק פעם אחת, וקל לחשב אותם.

חסרונות: 1) לא תמיד עובדת. 2) אי אפשר לקבוע מראש מה תהיה ה-target label (יש גירסא אחרת של השיטה שכן אפשר להחליט מראש).

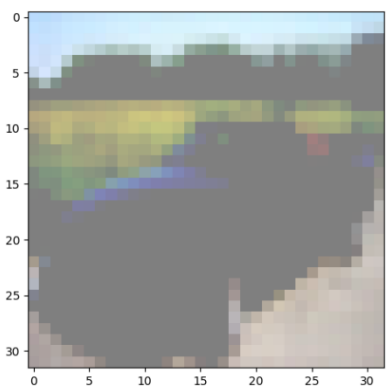
תוצאות:



Original image
*Tag: "**Car**"*



Noise



Adversarial image
*Predicted tag: "**Truck**"*