תרגיל בית 1 ברשתות ניורונים לתמונות

מגישים:

איתמר כץ 342689007

יונתן שליטא 318296217

Some explanation about the code:

The code has 3 files:

* Ex1.py – Main starting point. Here we have the training/validating code
* models.py – Contains all the different types of models we tried
* utils.py – Has helper functions such as for plotting, saving files etc.

To use the code one has to run it on the shell/cmd.

Here is a bottom-line example – the following code will run model 4 of question 1 with 15 epochs:

python .\Ex1.py --model-name "Q1\_4" --epochs 15

The rest of the flags that can be given to the code are less relevant and therefore we will not describe them here. You are welcome to run -h to see the usage of the code.

The model names are divided by Qx\_y where x is the question it is relevant for and y is the sub counter of the models.

The available models are:

* Q1\_1 up to Q1\_22 for models of question 1.
* Q2\_1 and Q2\_2 for models of question 2.

**שאלה 1**

We will now explain the different models and out conclusions. At the bottom we have a comparison of all the model plots and a list of the number of parameters to be learned. I urge you to see the models.py file to better understand the differences.

First, we will note the most important points, and then we will give the full list of the models.

Main point:

* Q1\_1 is the original net
* Q1\_4 had only 6,194 neurons and as can be seen in the plot the train loss could not go very low meaning it was a state of underfit.

Here is the full list of models and the reasoning of the changes:

* Q1\_1: This is just the original net. Total params: 62,006
* Q1\_2: In this net we removed a fully connected later. Total params: 37,406
* Q1\_3: In this net we changed the conv layers to have a kernel size of 3 instead of 5. Total params: 81,302
* Q1\_4: In this net we added 3 more layers of convolution, and since the output was small, we removed a fully connected. This resulted in a very low number of neurons which gave an underfitted net. Total params: 6,194
* Q1\_5: To have more neurons, instead of having conv layers with kernel 5x5, we have conv layers of kernel 3x3. this way we have 4 layers of conv (where applying pool after every two conv layers). This turned out to be negligible in comparison to the effect of the fully connected layers. Total params: 61,694
* Q1\_6: In this net we lowered the conv kernel to 3 but also added padding to the layers so it does not get smaller but do still apply pooling after each one. Total params: 44,028
* Q1\_7: This net is just like net 2 but with more neurons in the fully connected layers. Total params: 52,202
* Q1\_8: In this net we added channels to the conv layers (from 6 to 8 in conv1 and from 16 to 20 in conv2) resulting in a more complex net. Total params: 75,762
* Q1\_9: In this net we added channels to the conv layers (from 6 to 10 in conv1 and from 16 to 24 in conv2) resulting in a more complex net. Total params: 89,918
* Q1\_10: In this net we added channels to the conv layers (from 6 to 10 in conv1 and from 16 to 28 in conv2) resulting in a more complex net. Total params: 102,922
* Q1\_11: This net is designed to overfit by having 36 channels after the convolution layer which result in many neurons in the fully connected layers as well as an additional fully connected layer. Total params: 429,330
* Q1\_12: In continuation of Q1\_4: Q1\_4 was underfit due to so few neurons. Therefore, here we want to increase the number while leaving the conv layers. Hence, we will add padding to the conv layers but remove some of the channels. Total params: 44,912
* Q1\_13: To try to avoid overfitting, here we simply lower the number of channels of the last conv layer to 8. Total params: 36,798
* Q1\_14: to try to avoid overfitting, here we simply lower the number of channels of the last conv layer to 10. Total params: 43,100
* Q1\_15: to try to avoid overfitting, here we simply lower the number of channels of the last conv layer to 12. Total params: 49,402
* Q1\_16: to try to avoid overfitting, here we simply lower the number of channels of the last conv layer to 14. Total params: 55,704
* Q1\_17: We still have overfitting. So changed the conv layer to max of 6 channels and lowering the fully connected respectively Resulting in 30,044 parameters to learn. Total params: 30,044
* Q1\_18: We still have overfitting. So changed the conv layer to max of 5 channels and lowering the fully connected, respectively. Total params: 26,943
* Q1\_19: We now change the approach. Instead of lowering the number of neurons, maybe the right thing is to have more convolutional parameters as opposed to fully connected ones? Therefore, we added many convolutional layers which bump it up to 128 channels! This indeed gave an incredible result meaning we are getting closer. This net had a total of 550,570 parameters. Total params: 550,570
* Q1\_20: Continuing the idea of Q1\_19, but with less parameters - 356,810
* Q1\_21: Continuing the idea of Q1\_19, but with less parameters - 164,234
* Q1\_22: Continuing the idea of Q1\_19, but with less parameters - 105,258

כעט נסביר על נתוצאות שלנו.

**שאלה 2**

אחרי הורדת השכבות הליניאריות מהרשת, הביצועים של שרשת יורדים משמעותית – הגרף על ה loss לא יורד ולאומת הרשת המקורית, הaccuracy נמוך יותר כפי שניתן לראות בגרף הבא:

זה כמובן כתוצאה מכך שהרשת מוגבלת כעת ללמוד רק פונקציות ליניאריות.

השכבות הלא-ליניאריות ברשת היו:

* MaxPool2d
* F.relu

בנוסף, כאשר הוספנו את המימד של ה Fully Connected האחרון, המצב לא השתפר כפי שנתן לראות בגרף:

זאת כנראה מכיוון שהרשת כבר למדה את הגרף הליניארי המתאר בצורה הכי טובה את הקלט-פלט והוספת מימד בFully Connected לא שתנה את זה.

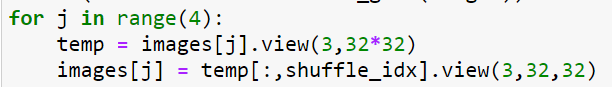
**שאלה 3**

אם ברצוננו לבחון את השפעת הסביבה הלוקאלית של הפיקסלים על זיהוי התמונה – לא מספיק להשתמש בפילטרים גדולים יותר, שכן גם פילטר גדול כולל בתוכו את הסביבה המקומית ועל כן היא עדיין תשפיע על אחוז הדיוק של הרשת. לכן, צריך להשוות למקרה בו הפילטר אינו כולל את הסביבה המקומית של הפיקסל – לדוגמא במצב בו מערבבים את הפיקסלים.

על מנת לבחון באופן בלתי תלוי את השפעת המיקום הלוקאלי מול הגלובאלי, הגדרנו פרמוטציה כללית

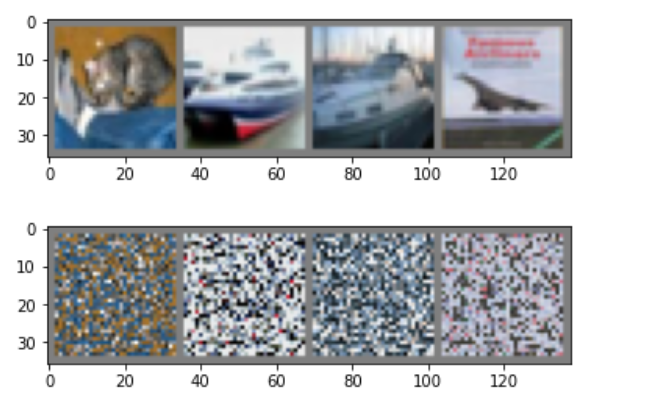


ובכל פעם לפני שהכנסנו תמונה לרשת סידרנו את הפיקסלים שלה מחדש ע"פ הפרמוטציה הנ"ל



הקפדנו לערבב את התמונה תוך שמירה על מימד הערוצים כשהיה, על מנת שלא ישתנו הצבעים של הפיקסלים עצמם.

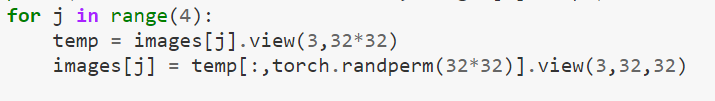
אכן השגנו פרמוטציה של התמונות:



כאשר הרצנו את הרשת על אותה פרמוטציה לכל התמונות, הגענו לאחוז דיוק של 40%, כלומר אבד 10% מהדיוק של הרשת. הרשת עדיין נעזרה במיקומים קבועים של פיקסלים בתמונה, אך לא נעזרה בסביבתם הקרובה, ועל כן איבדה מדיוקה.

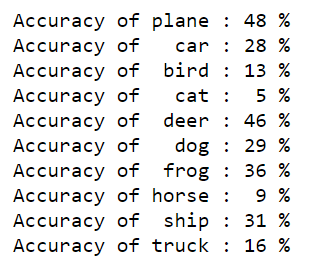
שאלה 4

כעת, במקום להשתמש בפרמוטציה קבועה לפיקסלים בכל תמונה, הגדרנו פרמוטציה רנדומאלית נפרדת לכל תמונה ותמונה. כלומר – כעת לרשת כבר אין אפשרות כלל ללמוד על מאפיינים של התמונה הנוגעים למיקומי הפיקסלים. התנאי:



כשהרצנו את הרשת גילינו שאחוז הדיוק שלה ירד ל25%. כלומר עדיין משמעותית טוב יותר מניחוש אקראי (דיוק של 10%).

ראינו גם שהמערכת ידעה לזהות מטוסים ואיילים בצורה טובה במיוחד, והתקשתה במיוחד לזהות חתולים.



ההשערה שלנו – היא שהסיבה שהמערכת עבדה יותר מניחוש אקראי, זה שהיא למדה להצביע על צבעים דומיננטים שמהווים פיצ'רים לכל קטגוריה (סוג הצבע, ומספר הפיקסלים בתמונה שצבועים בו). אולי לאור סיבה זו הרשת דווקא הצליחה לדוגמא לזהות מטוסים בצורה יחסית טובה (מטוסים לרב צבועים באותם צבעים – מטוס אפור ושמיים כחולים) ופחות הצליחה לזהות חתולים (שהצבעים שלהם הרבה פחות קבועים וצפויים משל מטוסים)