תרגיל בית 1 ברשתות ניורונים לתמונות

מגישים:

איתמר כץ 342689007

יונתן שליטא 318296217

For convenience, question 1 will be written in English and the rest in Hebrew.

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. <u>At the bottom we have a comparison of all the model plots and a list of the number of parameters to be learned</u>. 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.

- In **Q1_2-Q1_7** we tried general things to see what works.
- In **Q1_8-Q1_11** we tried different number of channels in the convolution layers to see if we can overfit the model.
- As a result of **Q1_11**, which made us believe we were overfitting, in **Q1_12-Q1_18** we try lowering the number of neurons.
- After **Q1_17-Q1_18** which were clearly underfit, in **Q1_19-Q1_25** we decided on going the other way around – meaning, trying to make the convolution layers have many channels in comparison to the fully connected layers thought by doing that, we still inevitably increase the fully connected layers, but by having only two of such layers we still reduce many of the weights. This gave us the best results we have.
- **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.
- **Q1_11** – fooled us since we thought it was overfit and we needed to decrease the number of neurons.
- **Q1_17-Q1_18** – Were clearly underfit
- **Q1_19** – Is probably the best result we got.
- **Q1_20- Q1_22** – Tried to have a bit less parameters
- **Q1_23- Q1_25** – Tried to have a bit more parameters

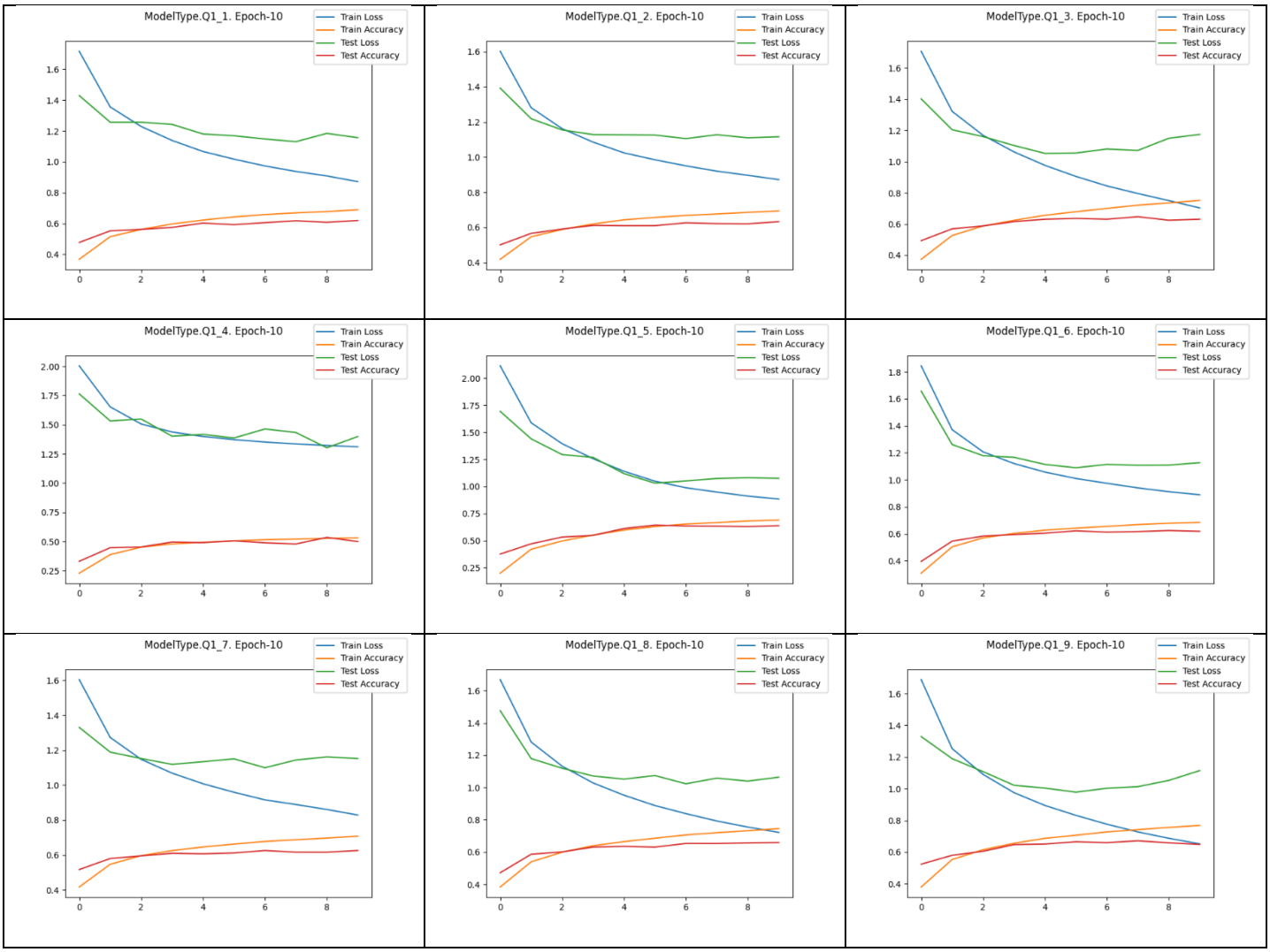
Here is the full list of models and the reasoning of the changes:

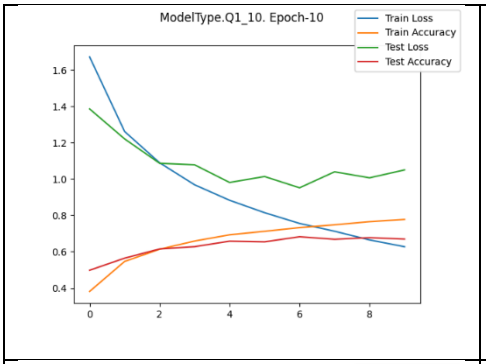
- Q1_1: This is just the original net. Total params: 62,006
- Q1_2: In this net we removed a fully connected later. Total params: 37,406
- Q1_3: In this net we changed the conv layers to have a kernel size of 3 instead of 5. Total params: 81,302
- Q1_4: In this net we added 3 more layers of convolution which forced us to remove a fully connected layer. This resulted in a very small number of neurons which gave an underfitted net. Total params: 6,194Q1_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

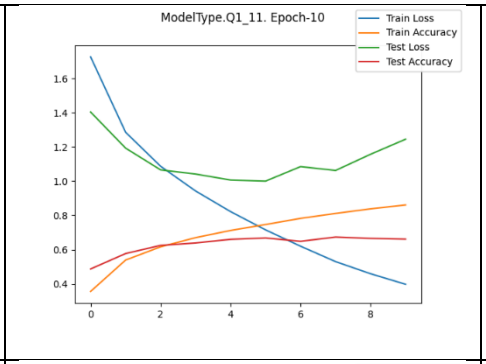
- Q1_7: This net is just like net Q1_2 but with more neurons in the fully connected layers. Total params: 52,202
- Q1_8: In this net we added channels to the conv layers (from 6 to 8 in conv1 and from 16 to 20 in conv2) resulting in a more complex net. Total params: 75,762
- Q1_9: In this net we added 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

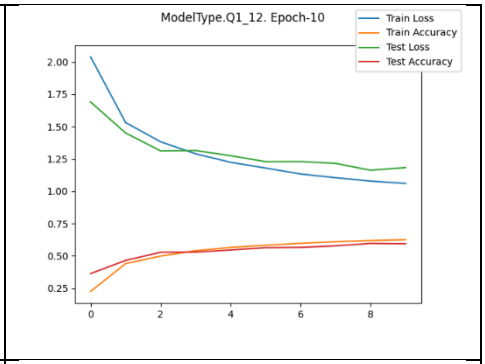
- Q1_21: Continuing the idea of Q1_19, but with less parameters - 164,234
- Q1_22: Continuing the idea of Q1_19, but with less parameters - 105,258
- Q1_23: Continuing the idea of Q1_19, but with more parameters - 960,298
- Q1_24: Continuing the idea of Q1_19, but with more parameters - 960,298
- Q1_25: Continuing the idea of Q1_19, but with more parameters - 2,141,610

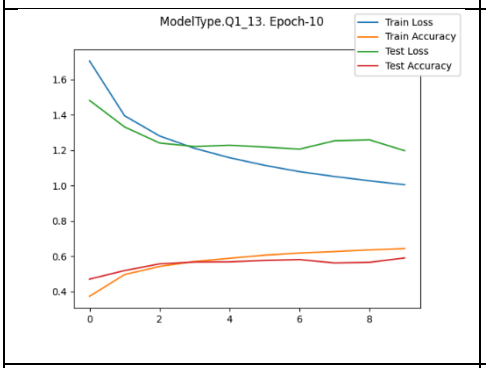
Following are the plots of the different models. In each plot, the title states which models is being seen and on how many epochs (the x axis represents the ephoc count). In addition, we show the loss and accuracy for both the training and validation.

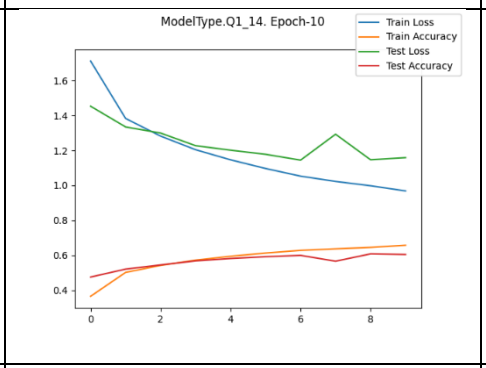


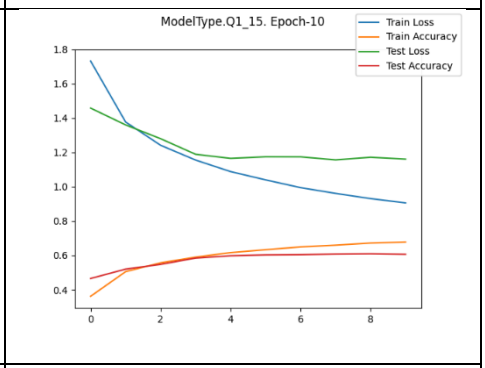

ModelType.Q1_10. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy

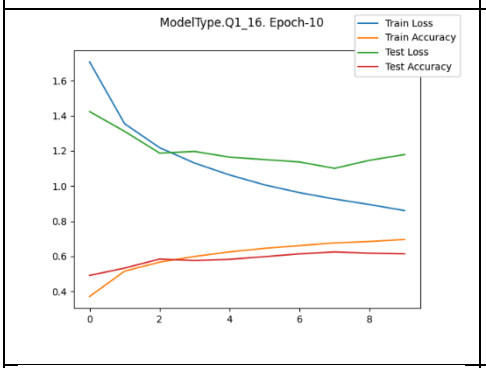

ModelType.Q1_11. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy

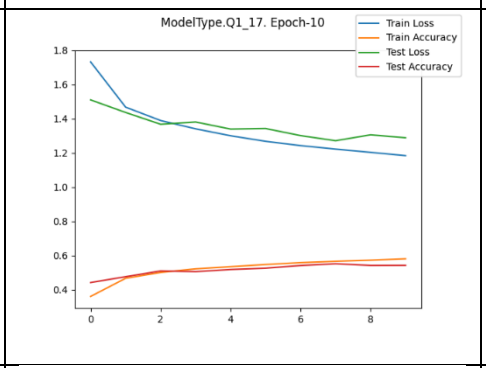

ModelType.Q1_12. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy

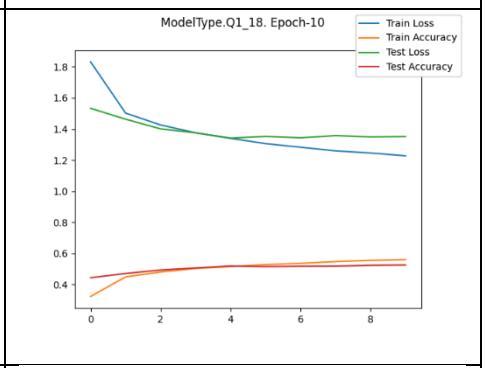

ModelType.Q1_13. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy

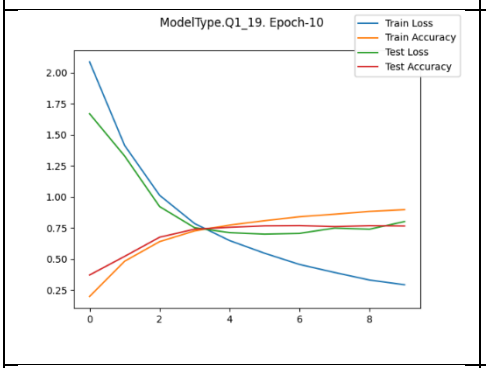

ModelType.Q1_14. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy

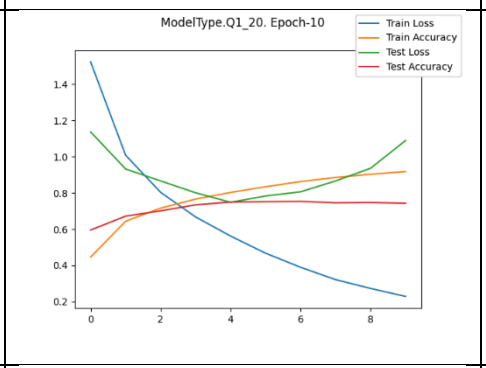

ModelType.Q1_15. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy

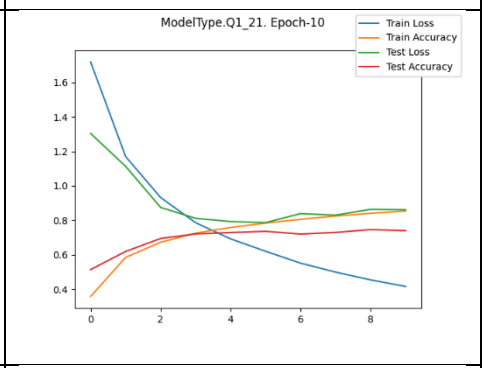

ModelType.Q1_16. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy

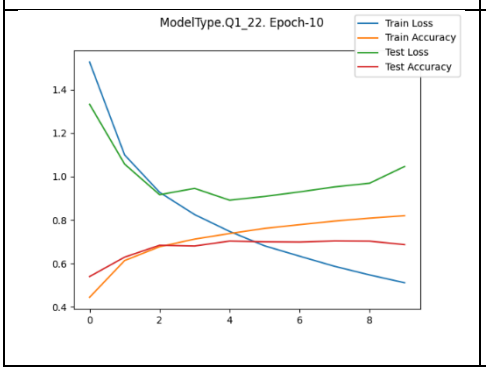

ModelType.Q1_17. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy

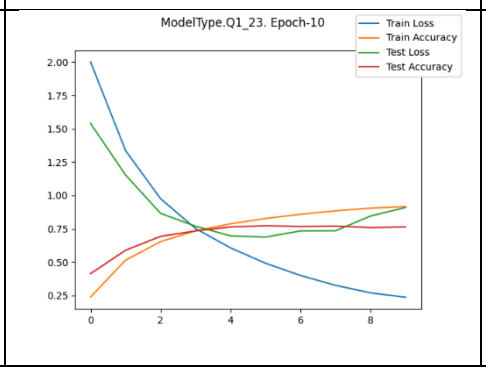

ModelType.Q1_18. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy

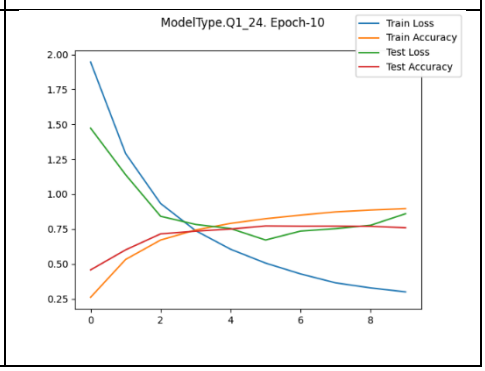

ModelType.Q1_19. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy

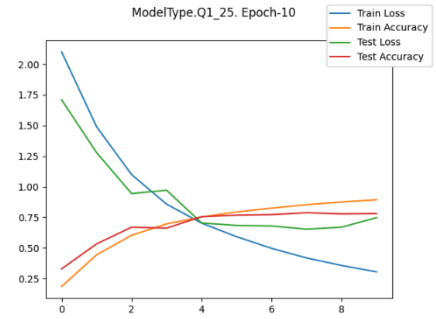

ModelType.Q1_20. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy


ModelType.Q1_21. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy


ModelType.Q1_22. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy


ModelType.Q1_23. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy


ModelType.Q1_24. Epoch-10
Train Loss
Train Accuracy
Test Loss
Test Accuracy

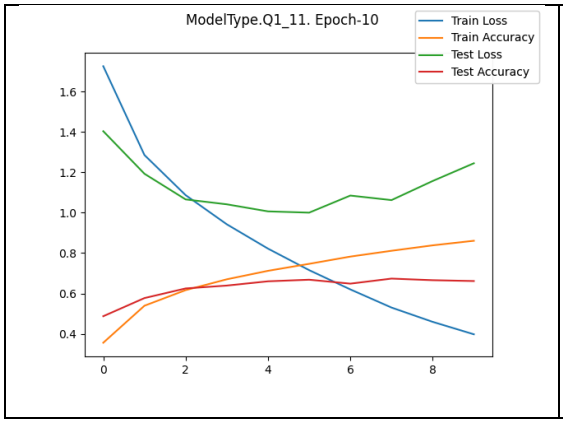
In the beginning we thought the layer was overfit because the training loss was becoming high after very fie epochs. This we eventually found to be a wrong assumption and in fact we could increase the convolutional channels and get a better result. Why is the training loss increasing after few epochs is not yet clear to us, but for sure we realized that by having many neurons while trying to keep the number of neurons from convolution close to the number of neurons of the fully connected layers gives a clear improvement on the testing accuracy.

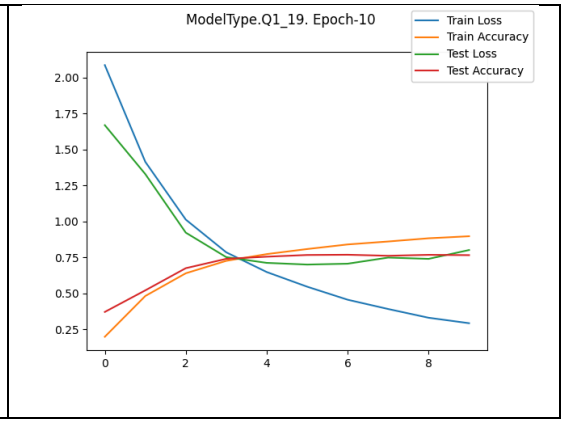
This can be seen in the comparison of Q1_19 and Q1_11 net.

Following is the summary of Q1_19:

Layer (type)	Output Shape	Number of Params
Conv2d-1	[-1, 32, 32, 32]	896
Conv2d-2	[-1, 32, 32, 32]	9,248
MaxPool2d-3	[-1, 32, 16, 16]	0
Conv2d-4	[-1, 64, 16, 16]	18,496
Conv2d-5	[-1, 64, 16, 16]	36,928
MaxPool2d-6	[-1, 64, 8, 8]	0
Conv2d-7	[-1, 128, 8, 8]	73,856
Conv2d-8	[-1, 128, 8, 8]	**147,584**
MaxPool2d-9	[-1, 128, 4, 4]	0
Linear-10	[-1, 128]	**262,272**
Linear-11	[-1, 10]	1,290

In this network (model Q1_19), the total params was 550,570, while in model Q1_11, the total params count was 429,330 which is less, yet its performance is worse and seems to be overfit as the test loss goes up very fast! (see figure)





Following is the summary of Q1_11:

Conv2d-1	[-1, 10, 28, 28]	760
MaxPool2d-2	[-1, 10, 14, 14]	0
Conv2d-3	[-1, 36, 10, 10]	9,036
MaxPool2d-4	[-1, 36, 5, 5]	0
Linear-5	[-1, 400]	**360,400**
Linear-6	[-1, 120]	48,120
Linear-7	[-1, 84]	10,164
Linear-8	[-1, 10]	850

This made us realize that having many fully connected layers as in Q1_11 can overfit the model (which is what made us believe we needed less parameters!) while having even more parameters in the convolutional layers might not.

In contrast, models Q1_4, Q1_17, Q1_18 seem to be underfit as the training loss does not go down (even after many more epoch. To show all the plots with the same number of epochs we chose to use 10, but most nets we run over around 50 epochs)

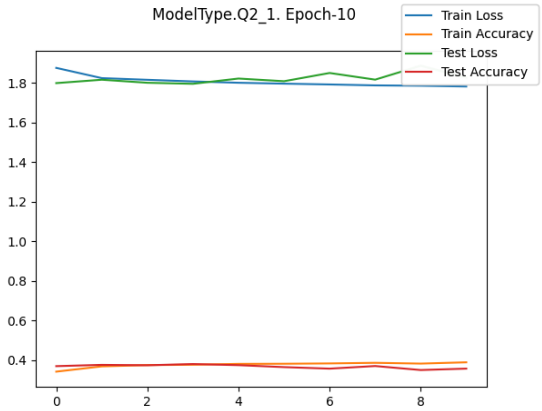
Finally, the chart with all the param count to be learned for each model:

Q1_1: Total params: 62,006	Q1_9: Total params: 89,918	Q1_16: Total params: 55,704
Q1_2: Total params: 37,406	Q1_10: Total params: 102,922	Q1_17: Total params: 30,044
Q1_3: Total params: 81,302	Q1_11: Total params: 429,330	Q1_18: Total params: 26,943
Q1_4: Total params: 6,194	Q1_12: Total params: 44,912	Q1_19: Total params: 550,570
Q1_5: Total params: 61,694	Q1_13: Total params: 36,798	Q1_20: Total params: 356,810
Q1_6: Total params: 44,028	Q1_14: Total params: 43,100	Q1_22: Total params: 105,258

Q1_7: Total params: 52,202 Q1_8: Total params: 75,762	Q1_15: Total params: 49,402	Q1_21: Total params: 164,234

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 (מודל Q2_1):



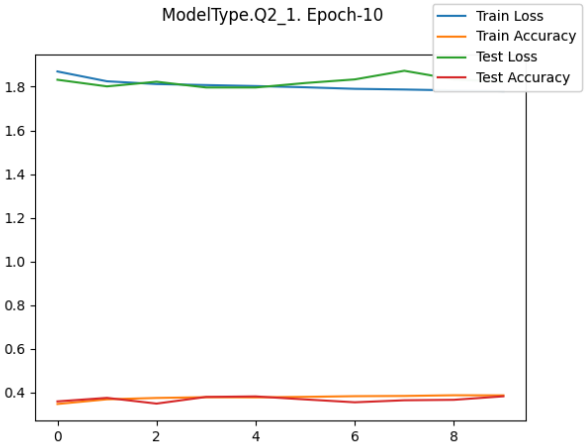
זה כמובן כתוצאה מכך שהרשת מוגבלת כעת ללמוד רק פונקציות ליניאריות.

בנוסף, מספר הניאורונים גדל מאוד שכן הורדנו את הMaxPool2d שהוריד את המימד לפני ה Fully Connected

השכבות הלא-ליניאריות ברשת היו:

- MaxPool2d
- F.relu

בנוסף, כאשר הוספנו את המימד של ה Fully Connected, כך שהגענו ל 2,618,562 פרמטרים, המצב לא השתפר כפי שנתן לראות בגרף הבא (מודל Q2_2):



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

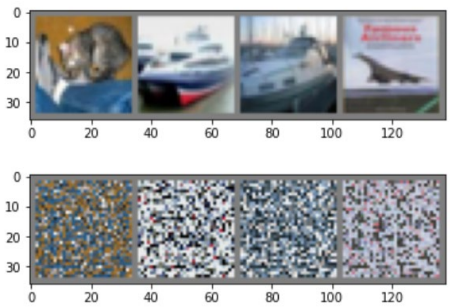
```
shuffle_idx = torch.randperm(32*32)
```

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

```
for j in range(4):
    temp = images[j].view(3,32*32)
    images[j] = temp[:,shuffle_idx].view(3,32,32)
```

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:

```
for j in range(4):
    temp = images[j].view(3,32*32)
    images[j] = temp[:,torch.randperm(32*32)].view(3,32,32)
```

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.

```
Accuracy of plane : 48 %
Accuracy of   car : 28 %
Accuracy of  bird : 13 %
Accuracy of   cat :  5 %
Accuracy of  deer : 46 %
Accuracy of   dog : 29 %
Accuracy of  frog : 36 %
Accuracy of horse :  9 %
Accuracy of  ship : 31 %
Accuracy of truck : 16 %
```

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)

שאלות תיאורטיות:

1. נראה תחילה כי ניתן לייצג את x(t) באופן הבא:

$$x(t) = \sum_{m\epsilon\mathbb{Z}} \delta(t-m) * x(m)$$

לכן:

$$L[x(t)] = L\left[\sum_{m\epsilon\mathbb{Z}} \delta(t-m) * x(m)\right]$$

מאחר ו-L ליניארית, ופונקציה של t ((x(m פרמטר לכל m):

$$L[x(t)] = \sum_{m\epsilon\mathbb{Z}} x(m) * L[\delta(t-m)]$$

נגדיר כעת:

$$h(t) = L[\delta(t)]$$

כלומר נוכל לייצג אותו באופן הבא:

$$h(t-m) = L[\delta(t-m)]$$

$$\sum_{m\epsilon\mathbb{Z}} x(m) * L[\delta(t-m)] = \sum_{m\epsilon\mathbb{Z}} x(m) * h(t-m) =$$

שזה יצוג של קונבולוצה של פונקציות h ו- x. כלומר L מתאר קונבולוציה

2. כאשר עושים סידור מחדש לactivation map, אין כלל חשיבות לסדר שבו ממסדרים את הקלטים לשכבת ה-FC. מאחר ובשכבת FC הארכיטקטורה זהה לכל ניורון (שמחובר לכל הניורונים בשכבה הבאה) וכמו כן המשקולות מאותחלות בערכים אקראיים, כך שקיימת סימטריות מוחלטת בתחילת הריצה בין מיקומי הניורונים.
3. א. נראה כי ReLU אינה ליניארית, ולכן אינה LTI: עבור קבוע חיובי s, נראה כי אינה סגורה בכפל לסקלר:

$$= ReLU(-1s) = 0 \neq -1\big(ReLU(s)\big) = -s$$

ב. נראה שה- strided pooling layer היא LTI. נתחיל בלהראות שהיא ליניארית
יהי בלוק בגודל mXn, יהי q ערך הפיקסל הימני-עליון, נגדיר סקלר s, אם נכפיל את הבלוק ב-s, ונבצע את הפול נקבל את הפיקסל הימני עליון שערכו יהיה s*q. מנגד, אם נפעיל את הפול על הבלוק ורק אז נכפיל ב-s, עדיין נקבל s*q. כלומר הפונקציה סגורה לכפל בסקלר.

הפונקציה היא translation invariant - יהיה y(t) הפלט עבור הסיגנל x(t), נראה שאם נסיט את הקלט x(t+s), ניקח את הפיקסל הימני עליון בבלוק הs+t, וגם אם נתבונן בפלט המוסט y(s+t) הוא יהיה שווה לפיקסל הימני-עליון בבלוק ה s+t, כלומר הם זהים.