DL Minor-1 Report

Task: Implement a CNN and utilize the CIFAR-10 dataset for the analysis

Objectives:

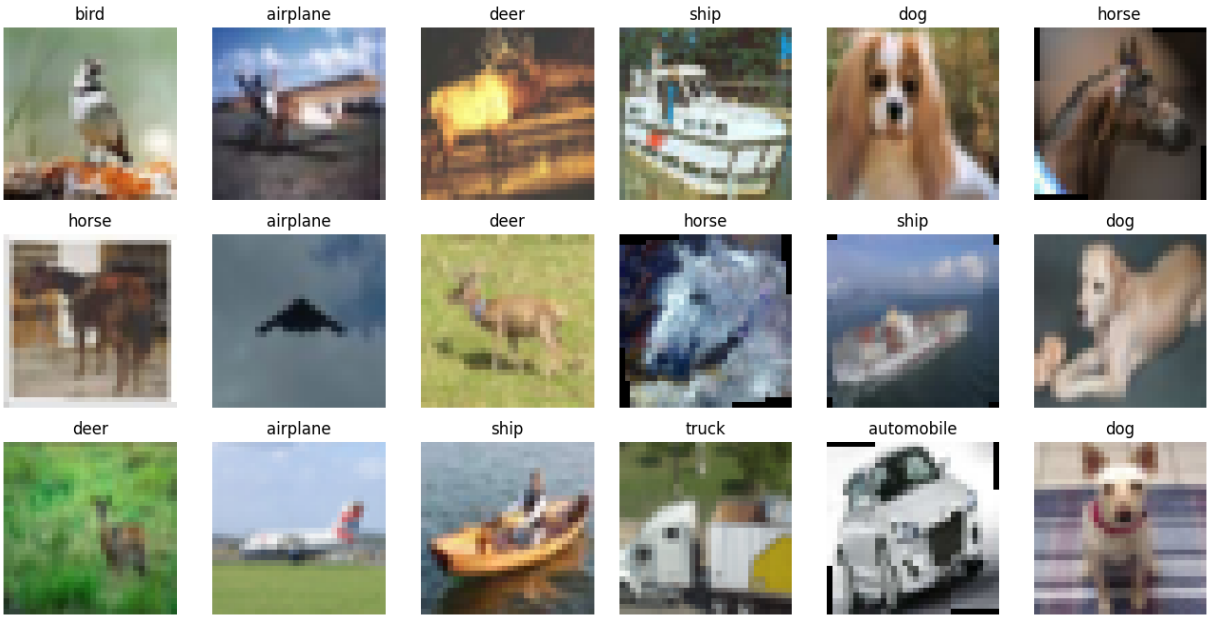
Question 1.

- As per directions in the question paper.
- Xavier initialization
- Rotation by +10/-10 degrees and gaussian noise.
- AvgPool
- Target 5 classes will be 0,2,4,6,8
- Architecture
    - Feature Extraction Layer
        - 6 Convolution Layer
        - 12 Filter
        - 1 Pooling Layer
    - Fully Connected Layer
        - 1024 nodes

Procedure:

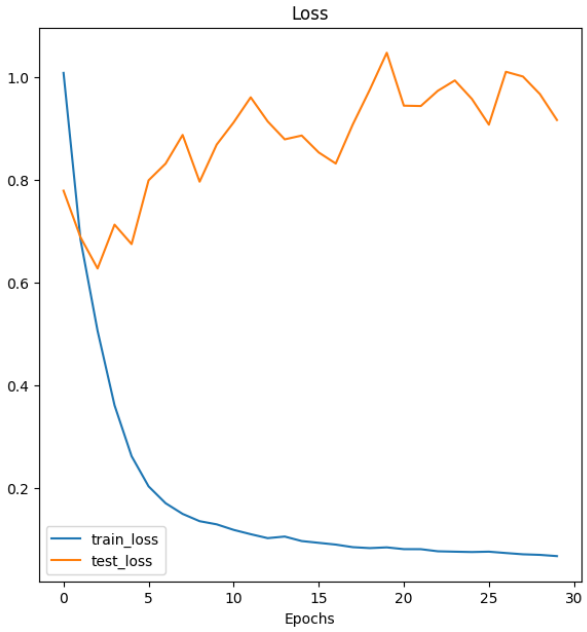
- Import required packages. (PyTorch, NumPy, Matplotlib, torchmetrics …etc.)
- Making compose transform to convert image to grayscale and to tensors.
- Download the Data with torchvision.dataset.CIFAR10() function with required parameters and transform as mentioned above.
- Visualizing random image samples from the train data set before and after transform.

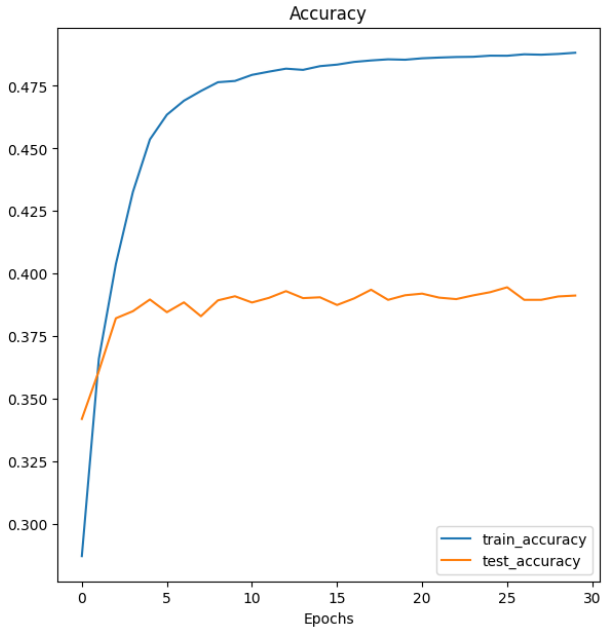
Original Images | Transformed Images



- Making subsets from complete data containing only required classes.
- Creating the data loaders for train and test datasets, respectively, with batch size 32.
- We are defining the PyTorch model class by inheriting nn.Module.
- Making model instance with appropriate arguments like activation functions.
- Defining the training step, testing step & train function that will be used for training the neural network.
- We‘ve defined one model, one with ReLU activation function.
- Training the model for 30 epochs with adam optimizer with learning rate 1e-3 and weight decay 1e-5, we get the results below.

```
 0%|          | 0/30 [00:00<?, ?it/s]
Epoch: 1 | train_loss: 1.0090 | train_acc: 0.2873 | test_loss: 0.7794 | test_acc: 0.3421
Epoch: 2 | train_loss: 0.6840 | train_acc: 0.3660 | test_loss: 0.6886 | test_acc: 0.3612
Epoch: 3 | train_loss: 0.5064 | train_acc: 0.4039 | test_loss: 0.6278 | test_acc: 0.3822
Epoch: 4 | train_loss: 0.3611 | train_acc: 0.4326 | test_loss: 0.7132 | test_acc: 0.3850
Epoch: 5 | train_loss: 0.2619 | train_acc: 0.4536 | test_loss: 0.6753 | test_acc: 0.3897
Epoch: 6 | train_loss: 0.2031 | train_acc: 0.4635 | test_loss: 0.7997 | test_acc: 0.3847
Epoch: 7 | train_loss: 0.1702 | train_acc: 0.4691 | test_loss: 0.8322 | test_acc: 0.3886
Epoch: 8 | train_loss: 0.1494 | train_acc: 0.4729 | test_loss: 0.8881 | test_acc: 0.3831
Epoch: 9 | train_loss: 0.1352 | train_acc: 0.4764 | test_loss: 0.7970 | test_acc: 0.3894
Epoch: 10 | train_loss: 0.1291 | train_acc: 0.4770 | test_loss: 0.8693 | test_acc: 0.3910
Epoch: 11 | train_loss: 0.1182 | train_acc: 0.4794 | test_loss: 0.9126 | test_acc: 0.3885
Epoch: 12 | train_loss: 0.1098 | train_acc: 0.4807 | test_loss: 0.9615 | test_acc: 0.3904
Epoch: 13 | train_loss: 0.1021 | train_acc: 0.4819 | test_loss: 0.9145 | test_acc: 0.3931
Epoch: 14 | train_loss: 0.1052 | train_acc: 0.4814 | test_loss: 0.8795 | test_acc: 0.3903
Epoch: 15 | train_loss: 0.0965 | train_acc: 0.4829 | test_loss: 0.8869 | test_acc: 0.3906
Epoch: 16 | train_loss: 0.0930 | train_acc: 0.4835 | test_loss: 0.8540 | test_acc: 0.3875
Epoch: 17 | train_loss: 0.0896 | train_acc: 0.4845 | test_loss: 0.8324 | test_acc: 0.3902
Epoch: 18 | train_loss: 0.0845 | train_acc: 0.4851 | test_loss: 0.9085 | test_acc: 0.3936
Epoch: 19 | train_loss: 0.0827 | train_acc: 0.4856 | test_loss: 0.9763 | test_acc: 0.3896
Epoch: 20 | train_loss: 0.0841 | train_acc: 0.4854 | test_loss: 1.0483 | test_acc: 0.3914
Epoch: 21 | train_loss: 0.0807 | train_acc: 0.4860 | test_loss: 0.9452 | test_acc: 0.3921
Epoch: 22 | train_loss: 0.0806 | train_acc: 0.4863 | test_loss: 0.9445 | test_acc: 0.3905
Epoch: 23 | train_loss: 0.0765 | train_acc: 0.4865 | test_loss: 0.9742 | test_acc: 0.3899
Epoch: 24 | train_loss: 0.0757 | train_acc: 0.4866 | test_loss: 0.9944 | test_acc: 0.3914
Epoch: 25 | train_loss: 0.0751 | train_acc: 0.4870 | test_loss: 0.9584 | test_acc: 0.3926
Epoch: 26 | train_loss: 0.0758 | train_acc: 0.4870 | test_loss: 0.9082 | test_acc: 0.3946
Epoch: 27 | train_loss: 0.0731 | train_acc: 0.4876 | test_loss: 1.0113 | test_acc: 0.3896
Epoch: 28 | train_loss: 0.0707 | train_acc: 0.4874 | test_loss: 1.0020 | test_acc: 0.3896
Epoch: 29 | train_loss: 0.0695 | train_acc: 0.4877 | test_loss: 0.9678 | test_acc: 0.3909
Epoch: 30 | train_loss: 0.0671 | train_acc: 0.4882 | test_loss: 0.9172 | test_acc: 0.3913
total training time: 491.051 sec.
```



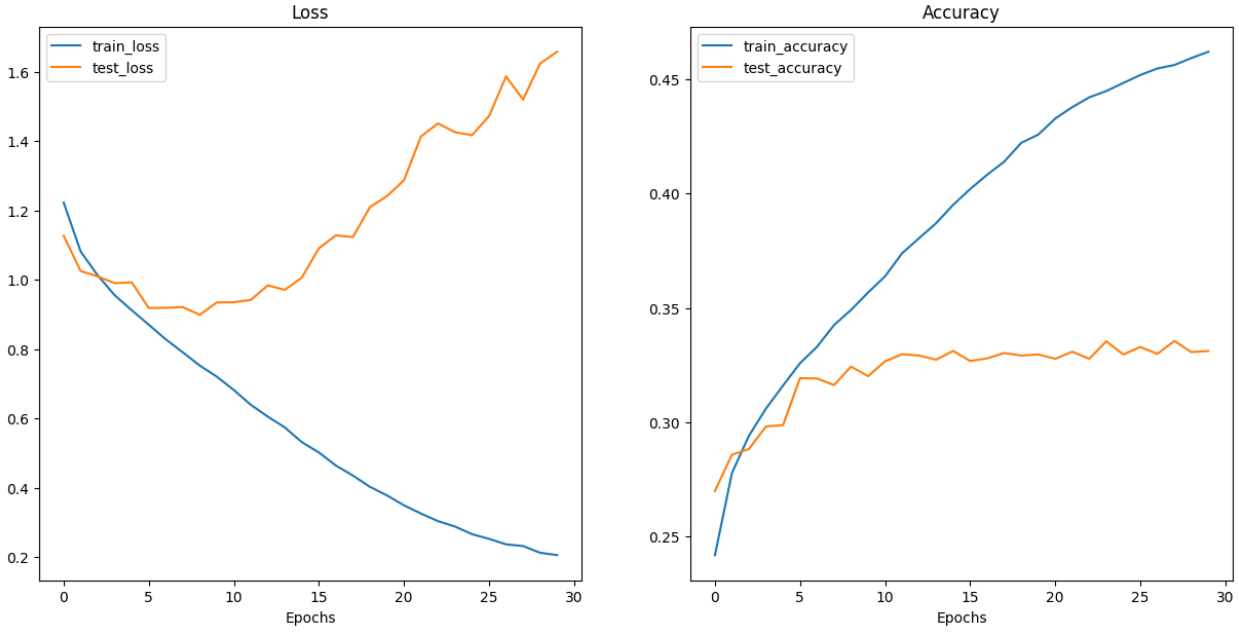


Observations:

- Seeing the curves above, we can see that the model is highly overfitting. We have tried to get better performance by applying dropouts, but it still needs to improve significantly.
- But why? The answer is simply because of the architecture we are following. The model can get much better; in contrast, we have done one other experiment i.e. training a simple neural network for classification without convolutions, and the results are shown below.

Epoch: 1 | train_loss: 1.2229 | train_acc: 0.2419 | test_loss: 1.1269 | test_acc: 0.2700
Epoch: 2 | train_loss: 1.0815 | train_acc: 0.2778 | test_loss: 1.0253 | test_acc: 0.2858
Epoch: 3 | train_loss: 1.0128 | train_acc: 0.2942 | test_loss: 1.0099 | test_acc: 0.2883
Epoch: 4 | train_loss: 0.9555 | train_acc: 0.3060 | test_loss: 0.9903 | test_acc: 0.2982
Epoch: 5 | train_loss: 0.9122 | train_acc: 0.3161 | test_loss: 0.9925 | test_acc: 0.2988
Epoch: 6 | train_loss: 0.8705 | train_acc: 0.3258 | test_loss: 0.9187 | test_acc: 0.3193
Epoch: 7 | train_loss: 0.8283 | train_acc: 0.3330 | test_loss: 0.9192 | test_acc: 0.3192
Epoch: 8 | train_loss: 0.7909 | train_acc: 0.3425 | test_loss: 0.9213 | test_acc: 0.3163
Epoch: 9 | train_loss: 0.7526 | train_acc: 0.3492 | test_loss: 0.8988 | test_acc: 0.3244
Epoch: 10 | train_loss: 0.7206 | train_acc: 0.3568 | test_loss: 0.9345 | test_acc: 0.3202
Epoch: 11 | train_loss: 0.6822 | train_acc: 0.3638 | test_loss: 0.9354 | test_acc: 0.3267
Epoch: 12 | train_loss: 0.6394 | train_acc: 0.3739 | test_loss: 0.9420 | test_acc: 0.3298
Epoch: 13 | train_loss: 0.6053 | train_acc: 0.3805 | test_loss: 0.9839 | test_acc: 0.3292
Epoch: 14 | train_loss: 0.5741 | train_acc: 0.3871 | test_loss: 0.9710 | test_acc: 0.3274
Epoch: 15 | train_loss: 0.5316 | train_acc: 0.3951 | test_loss: 1.0064 | test_acc: 0.3312
Epoch: 16 | train_loss: 0.5019 | train_acc: 0.4020 | test_loss: 1.0908 | test_acc: 0.3268
Epoch: 17 | train_loss: 0.4642 | train_acc: 0.4083 | test_loss: 1.1282 | test_acc: 0.3279
Epoch: 18 | train_loss: 0.4354 | train_acc: 0.4140 | test_loss: 1.1233 | test_acc: 0.3303
Epoch: 19 | train_loss: 0.4029 | train_acc: 0.4222 | test_loss: 1.2097 | test_acc: 0.3292

Epoch: 20 | train_loss: 0.3781 | train_acc: 0.4258 | test_loss: 1.2410 | test_acc: 0.3297
Epoch: 21 | train_loss: 0.3495 | train_acc: 0.4328 | test_loss: 1.2873 | test_acc: 0.3278
Epoch: 22 | train_loss: 0.3258 | train_acc: 0.4378 | test_loss: 1.4131 | test_acc: 0.3309
Epoch: 23 | train_loss: 0.3038 | train_acc: 0.4421 | test_loss: 1.4511 | test_acc: 0.3278
Epoch: 24 | train_loss: 0.2882 | train_acc: 0.4449 | test_loss: 1.4259 | test_acc: 0.3354
Epoch: 25 | train_loss: 0.2662 | train_acc: 0.4484 | test_loss: 1.4171 | test_acc: 0.3297
Epoch: 26 | train_loss: 0.2527 | train_acc: 0.4519 | test_loss: 1.4719 | test_acc: 0.3329
Epoch: 27 | train_loss: 0.2370 | train_acc: 0.4547 | test_loss: 1.5867 | test_acc: 0.3299
Epoch: 28 | train_loss: 0.2320 | train_acc: 0.4562 | test_loss: 1.5199 | test_acc: 0.3356
Epoch: 29 | train_loss: 0.2126 | train_acc: 0.4593 | test_loss: 1.6236 | test_acc: 0.3307
Epoch: 30 | train_loss: 0.2058 | train_acc: 0.4620 | test_loss: 1.6578 | test_acc: 0.3312
total training time: 362.087 sec.



- We can see that our CNN is not performing that well in comparison.

Task: Implement an AutoEncoder + Classifier and utilize the CIFAR-10 dataset for the analysis

Objectives:

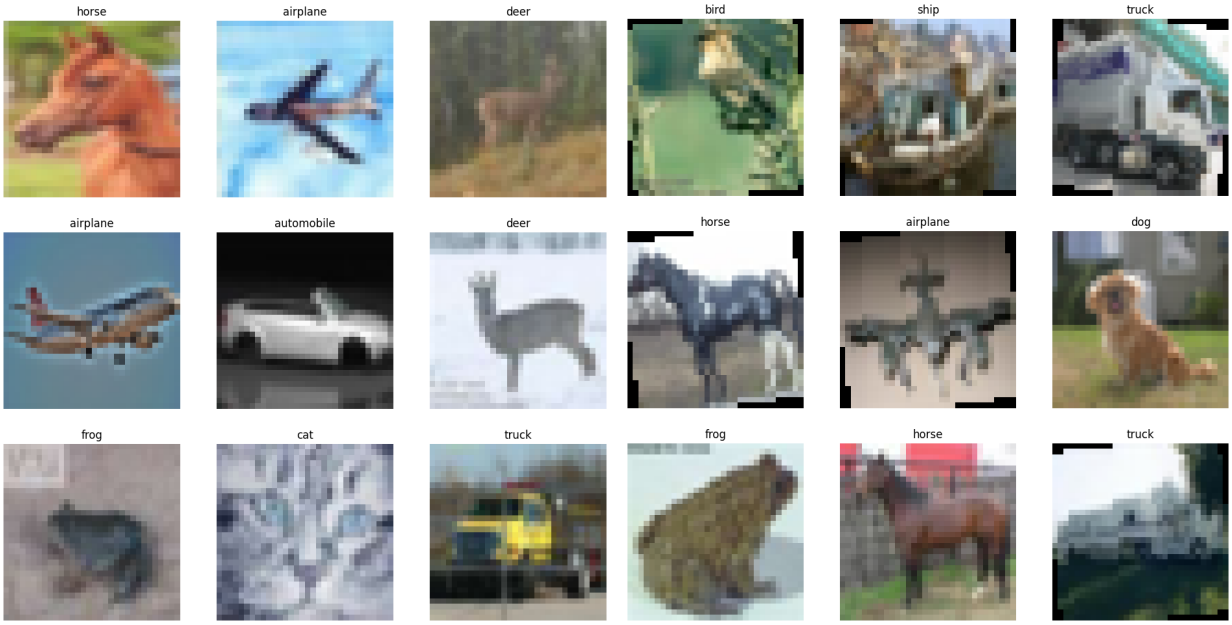
Question 2.

- As per directions in the question paper.
- Xavier initialization
- Rotation by +10/-10 degrees and gaussian noise.
- Target 5 classes will be 0,2,4,6,8
- Architecture
    - AutoEncoder
        - 3 layers
    - Fully Connected Layer
        - 512 nodes

Procedure:

- Import required packages. (PyTorch, NumPy, Matplotlib, torchmetrics …etc.)
- Making compose transform to convert image to grayscale and to tensors.
- Download the Data with torchvision.dataset.CIFAR10() function with required parameters and transform as mentioned above.
- Visualizing random image samples from the train data set before and after transform.

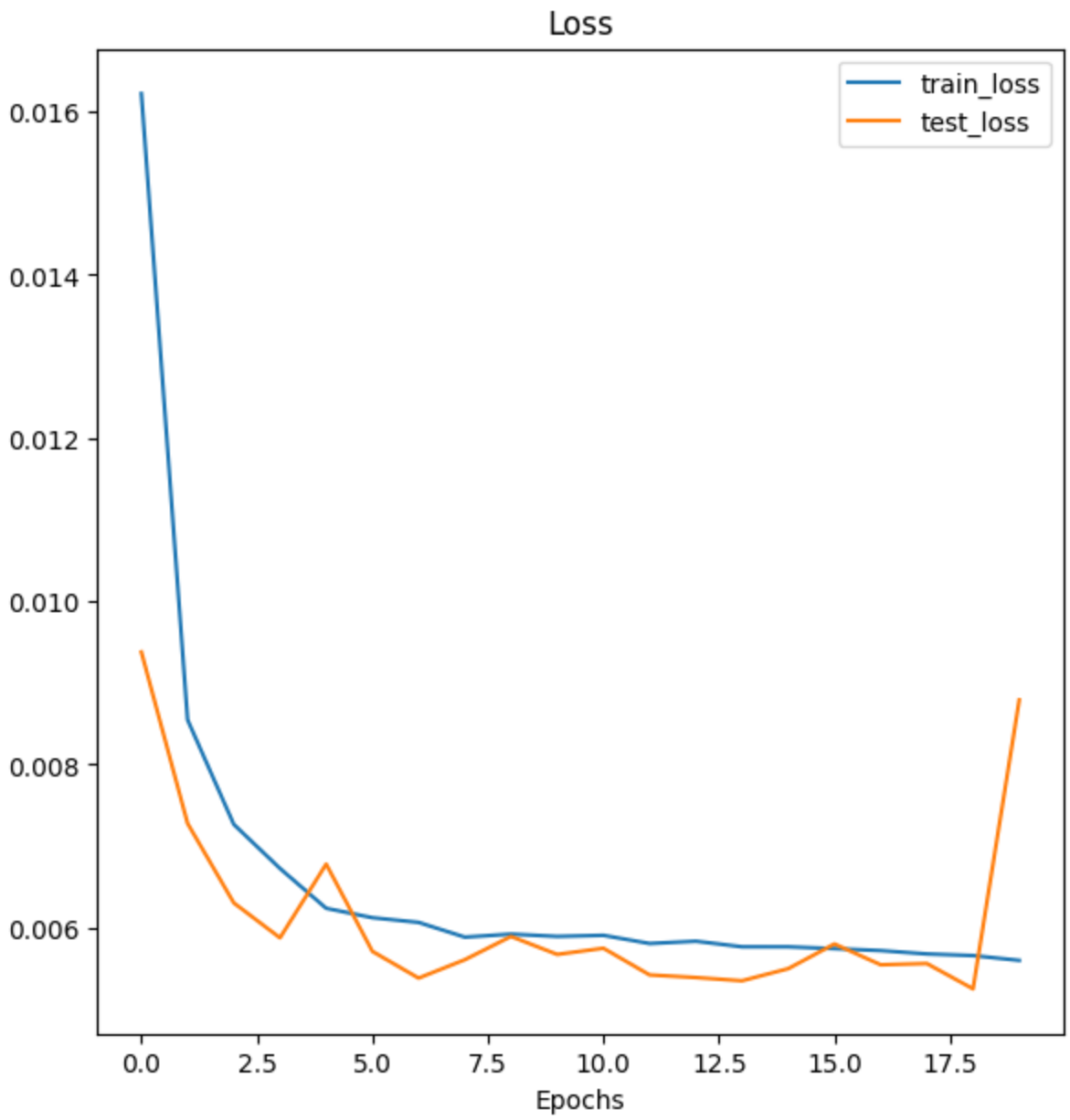
Original Images | Transformed Images



- Making subsets from complete data containing only required classes.
- Creating the data loaders for train and test datasets, respectively, with batch size 32.
- We are defining the PyTorch model class by inheriting nn.Module.
- Making model instance with appropriate arguments like activation functions.
- Defining the training step, testing step & train function that will be used for training the neural network.

- We‘ve defined one model, one with the ReLU activation function.
- Training the model for 20 epochs with adam optimizer with learning rate 1e-3, we get the results below.

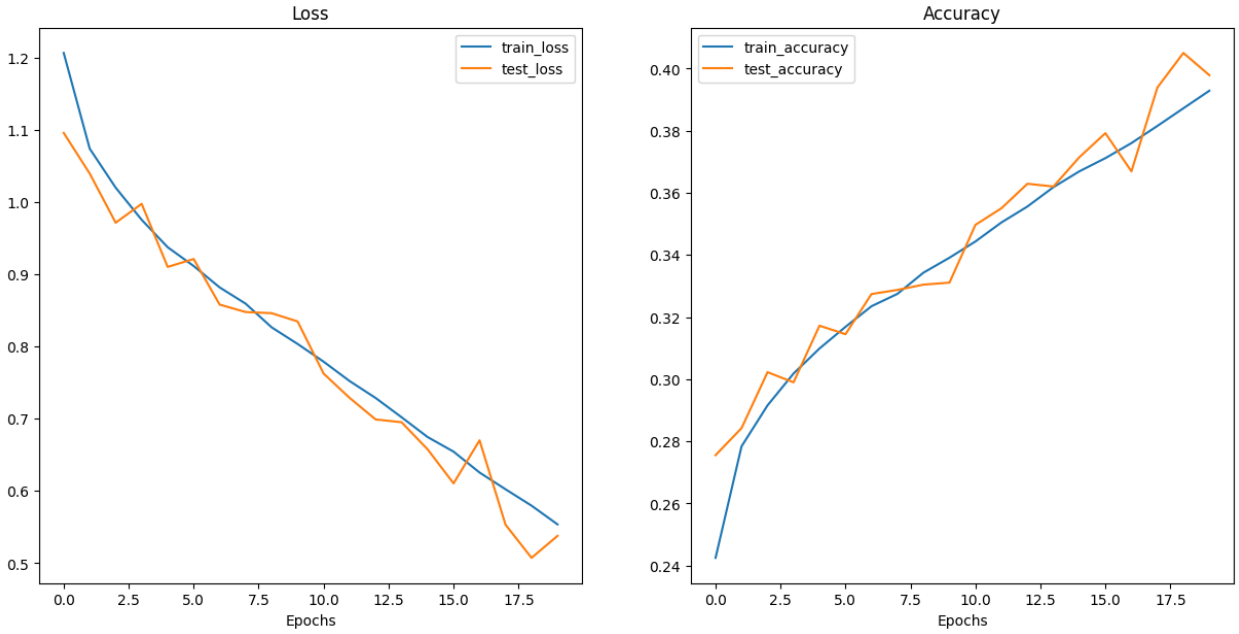
```
  0%|          | 0/20 [00:00<?, ?it/s]
EPOCH: 0 | Train loss: 0.01622 | Test loss: 0.00938
EPOCH: 1 | Train loss: 0.00855 | Test loss: 0.00728
EPOCH: 2 | Train loss: 0.00727 | Test loss: 0.00631
EPOCH: 3 | Train loss: 0.00673 | Test loss: 0.00588
EPOCH: 4 | Train loss: 0.00624 | Test loss: 0.00678
EPOCH: 5 | Train loss: 0.00612 | Test loss: 0.00571
EPOCH: 6 | Train loss: 0.00607 | Test loss: 0.00538
EPOCH: 7 | Train loss: 0.00589 | Test loss: 0.00561
EPOCH: 8 | Train loss: 0.00592 | Test loss: 0.00589
EPOCH: 9 | Train loss: 0.00589 | Test loss: 0.00567
EPOCH: 10 | Train loss: 0.00591 | Test loss: 0.00575
EPOCH: 11 | Train loss: 0.00581 | Test loss: 0.00542
EPOCH: 12 | Train loss: 0.00584 | Test loss: 0.00539
EPOCH: 13 | Train loss: 0.00577 | Test loss: 0.00535
EPOCH: 14 | Train loss: 0.00577 | Test loss: 0.00550
EPOCH: 15 | Train loss: 0.00575 | Test loss: 0.00580
EPOCH: 16 | Train loss: 0.00572 | Test loss: 0.00555
EPOCH: 17 | Train loss: 0.00568 | Test loss: 0.00556
EPOCH: 18 | Train loss: 0.00566 | Test loss: 0.00525
EPOCH: 19 | Train loss: 0.00560 | Test loss: 0.00879
total training time: 126.575 sec.
```



- Now Training the Classifier, with one fully connected layer with 512 nodes.
- Using Adam optimizer with learning rate 10e-3, we get the results.

0%| | 0/20 [00:00<?, ?it/s]
Epoch: 1 | train_loss: 1.2065 | train_acc: 0.2425 | test_loss: 1.0958 | test_acc: 0.2755
Epoch: 2 | train_loss: 1.0740 | train_acc: 0.2784 | test_loss: 1.0396 | test_acc: 0.2842
Epoch: 3 | train_loss: 1.0200 | train_acc: 0.2916 | test_loss: 0.9715 | test_acc: 0.3023
Epoch: 4 | train_loss: 0.9755 | train_acc: 0.3018 | test_loss: 0.9976 | test_acc: 0.2990
Epoch: 5 | train_loss: 0.9375 | train_acc: 0.3099 | test_loss: 0.9103 | test_acc: 0.3172
Epoch: 6 | train_loss: 0.9115 | train_acc: 0.3169 | test_loss: 0.9212 | test_acc: 0.3145
Epoch: 7 | train_loss: 0.8819 | train_acc: 0.3235 | test_loss: 0.8581 | test_acc: 0.3274
Epoch: 8 | train_loss: 0.8594 | train_acc: 0.3275 | test_loss: 0.8478 | test_acc: 0.3288
Epoch: 9 | train_loss: 0.8266 | train_acc: 0.3343 | test_loss: 0.8462 | test_acc: 0.3305

Epoch: 10 | train_loss: 0.8036 | train_acc: 0.3391 | test_loss: 0.8348 | test_acc: 0.3311
Epoch: 11 | train_loss: 0.7791 | train_acc: 0.3444 | test_loss: 0.7626 | test_acc: 0.3497
Epoch: 12 | train_loss: 0.7523 | train_acc: 0.3505 | test_loss: 0.7290 | test_acc: 0.3551
Epoch: 13 | train_loss: 0.7289 | train_acc: 0.3556 | test_loss: 0.6991 | test_acc: 0.3629
Epoch: 14 | train_loss: 0.7021 | train_acc: 0.3618 | test_loss: 0.6952 | test_acc: 0.3620
Epoch: 15 | train_loss: 0.6748 | train_acc: 0.3669 | test_loss: 0.6577 | test_acc: 0.3715
Epoch: 16 | train_loss: 0.6545 | train_acc: 0.3712 | test_loss: 0.6106 | test_acc: 0.3792
Epoch: 17 | train_loss: 0.6256 | train_acc: 0.3760 | test_loss: 0.6703 | test_acc: 0.3669
Epoch: 18 | train_loss: 0.6024 | train_acc: 0.3816 | test_loss: 0.5534 | test_acc: 0.3939
Epoch: 19 | train_loss: 0.5798 | train_acc: 0.3872 | test_loss: 0.5075 | test_acc: 0.4051
Epoch: 20 | train_loss: 0.5538 | train_acc: 0.3929 | test_loss: 0.5379 | test_acc: 0.3979
total training time: 175.097 sec.



Observations:

- The curves above show that the model performs well and can be optimized further to get better results.
- The model is neither overfitting nor underfitting.

Comparison:

- From the above experiments, we can see that the auto-encoded classifier performs similarly to cnn based model.
- The auto-encoded classifier can even surpass the performance of cnn in our settings because of the architectural limitations we are applying on the CNN.
- If we follow betters architecture for the cnn, the cnns will outperform the autoencoders.

References:

https://cs230.stanford.edu/section/4/
https://poloclub.github.io/cnn-explainer/
https://pytorch.org/docs/stable/data.html
https://pytorch.org/vision/stable/transforms.html
https://pytorch.org/docs/stable/index.html
https://numpy.org/doc/stable/reference/index.html
https://torchmetrics.readthedocs.io/en/stable/
https://stackoverflow.com/
Class ppts.

Links:

Question 1:-
https://colab.research.google.com/drive/1eLBTaE3audGTk83WvfzM5TIkObaVCaCO?usp=sharing

Question 2:-
https://colab.research.google.com/drive/1A4yyxYGONwz-rraaw9pTfHz-oqvVEtfU?usp=sharing