DLOps Report

Assignment 1

Ayush Abrol

B20AI052

Question 01

Aim: Train ResNet18 on Tiny ImageNet dataset with X=SGD (B20AI052) as the optimizer for classification task. Plot curves for training loss, training accuracy, validation accuracy and report the final test accuracy. Here consider accuracy as top-5 accuracy.

1. Use CrossEntropy as the final classification loss function.

2. Use Triplet Loss with hard mining as the final classification loss function.

3. Use Central Loss as the final classification loss function.

Compare the performance of different models.

Dataset Link: http://cs231n.stanford.edu/tiny-imagenet-200.zip

Procedure:

- All the necessary dependencies and libraries were taken care of initially like:
    - numpy, pandas, matplotlib for data manipulation and visualization.
    - Torch, torchvision, torch.nn, PIL for for implementing Deep Learning architectures where necessary using PyTocrch.
- Loading of the training and the testing TinyImageNet dataset and normalizing it using torchvision's transforms module. **100k training samples, 10k validation and 10k testing samples** are loaded. Note: **Only train samples and validation samples have the annotations for their labels and testing samples have no annotations provided in the dataset for the labels.**
- Total number of classes = 200
- Read the annotations for the training and validation data using file methods.

```
array([['n00001740', 'entity'],
       ['n00001930', 'physical entity'],
       ['n00002137', 'abstraction, abstract entity'],
       ...,
       ['n15299585', 'usance'],
       ['n15299783', 'window'],
       ['n15300051', '9/11, 9-11, September 11, Sept. 11, Sep 11']],
      dtype='<U371')
```

Train labels and their annotations.

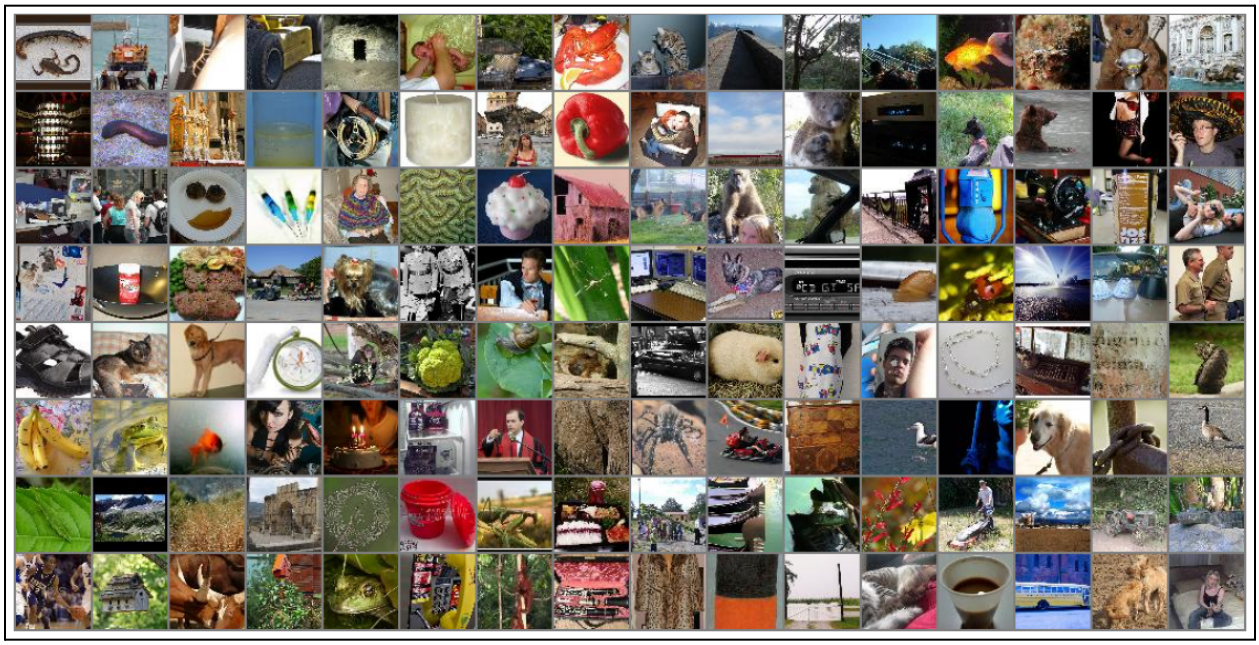
```
array([['val_0.JPEG', 'n03444034', '0', '32', '44', '62'],
       ['val_1.JPEG', 'n04067472', '52', '55', '57', '59'],
       ['val_2.JPEG', 'n04070727', '4', '0', '60', '55'],
       ...,
       ['val_9997.JPEG', 'n03424325', '10', '10', '38', '42'],
       ['val_9998.JPEG', 'n01629819', '28', '18', '63', '31'],
       ['val_9999.JPEG', 'n02699494', '17', '33', '28', '39']],
      dtype='<U13')
```

Validation labels and their annotations.

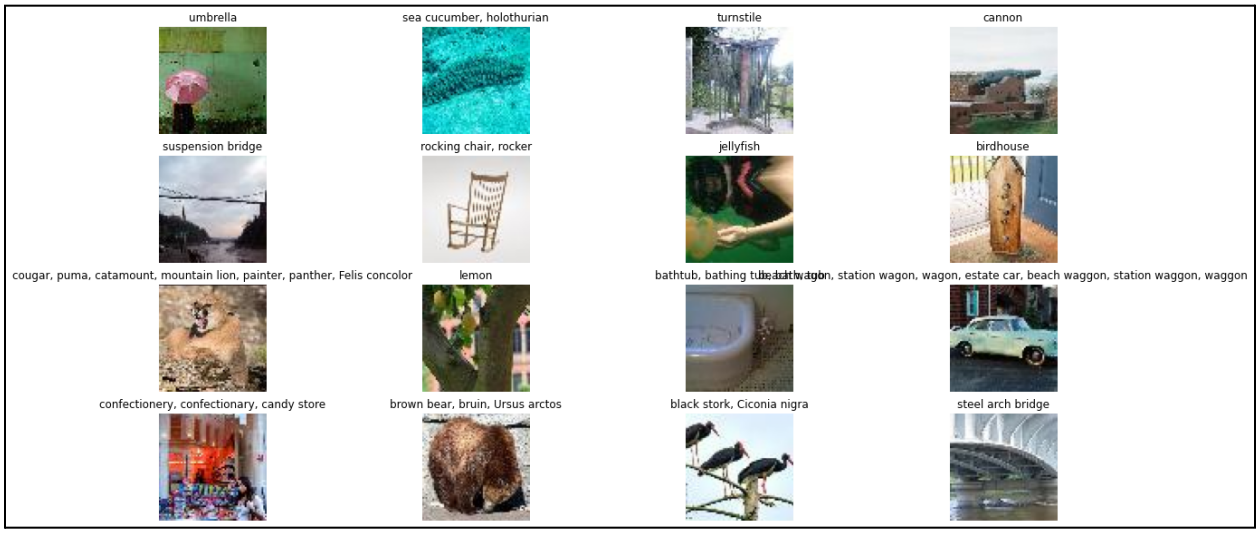
- Dataset size and batch size:

```
Number of images in the training set:  100000
Number of images in the validation set:  5000
Number of images in the test set:  5000
Shape of the one image batch:  torch.Size([128, 3, 64, 64])
Shape of the one label batch:  torch.Size([128])
```

- Visualization of the dataset



- Some samples images with their labels.



- Then, I defined the Basic Block class and ResNet class.
- The BasicBlock class defines the building block for the ResNet model. It consists of two convolutional layers with batch normalization and ReLU activation functions followed by a shortcut connection. The shortcut connection is used to add the

original input tensor to the output tensor of the two convolutional layers, thus making the model easier to optimize.

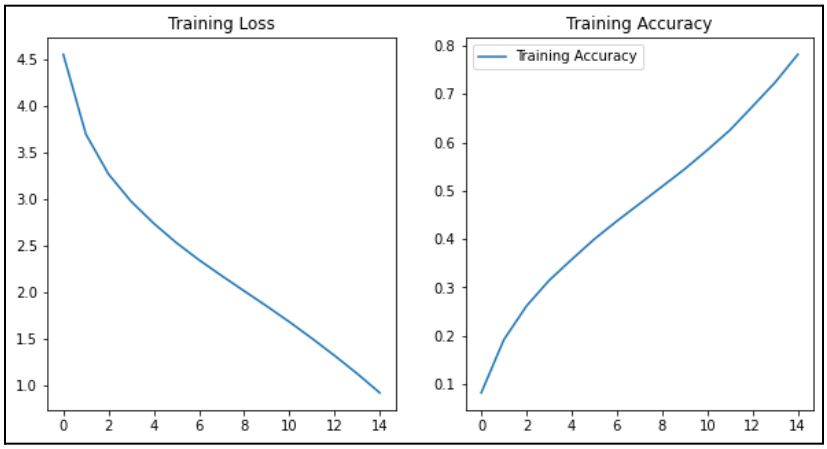
- The ResNet class defines the ResNet model architecture using the BasicBlock building block. The architecture consists of a 3x3 convolutional layer with batch normalization and ReLU activation, followed by four layers of BasicBlock modules. The number of BasicBlock modules in each layer is specified by the num_blocks argument. The output of the final BasicBlock layer is fed into an average pooling layer, which reduces the spatial dimensions of the feature map. The output of the average pooling layer is then flattened and passed through a linear layer to produce the final output classification.
- The architecture is flexible and can be modified by adjusting the number of BasicBlock modules in each layer and the number of output classes in the final linear layer.

- Then, instantiated the ResNet18 object with 200 output classes and 4 blocks each of size 2. It is such that [2, 2, 2, 2].
- Used the Stochastic Gradient Descent optimizer with model ResNet18 parameters, learning rate=1e-3 and momentum=0.9.
- Created a function for calculating the CrossEntropyLoss between outputs obtained the ground truth values.
- Then, trained the model for 15 epochs with CrossEntropy loss as the defined loss.

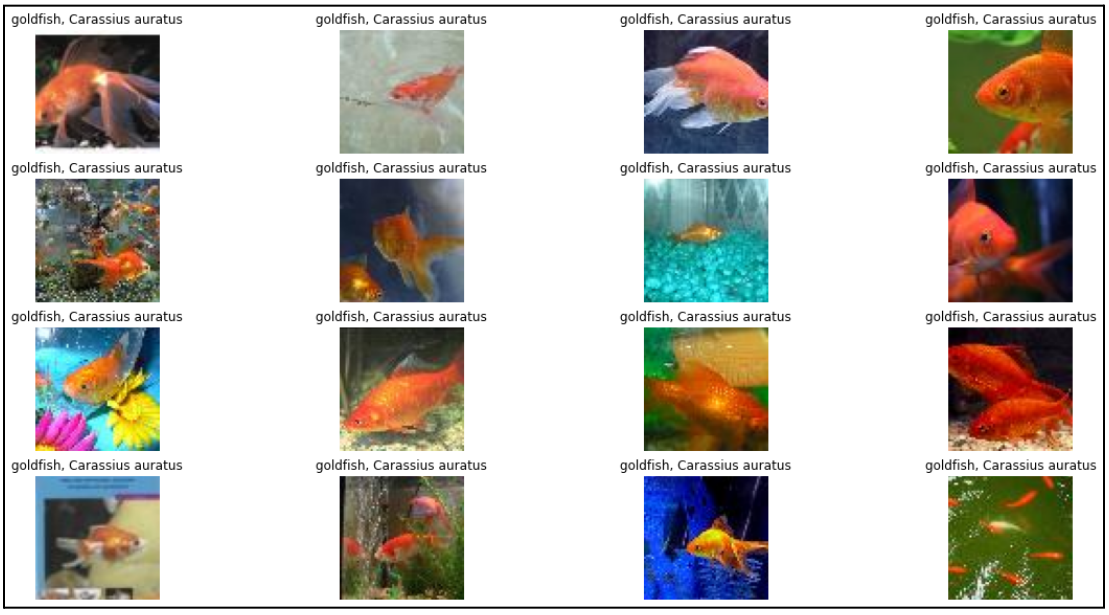
```
Epoch: 1 Training Loss:  4.547990052901265      Training Accuracy:  0.08184
Epoch: 2 Training Loss:  3.6924747981683677     Training Accuracy:  0.19174
Epoch: 3 Training Loss:  3.264556926839492      Training Accuracy:  0.26147
Epoch: 4 Training Loss:  2.9719543185685295     Training Accuracy:  0.31442
Epoch: 5 Training Loss:  2.7366216679668183     Training Accuracy:  0.35783
Epoch: 6 Training Loss:  2.5285627396820147     Training Accuracy:  0.39979
Epoch: 7 Training Loss:  2.3448896081856145     Training Accuracy:  0.43742
Epoch: 8 Training Loss:  2.175702697175848      Training Accuracy:  0.47299
Epoch: 9 Training Loss:  2.0116189838675282     Training Accuracy:  0.50885
Epoch: 10 Training Loss:  1.8483535475133326    Training Accuracy:  0.54491
```

```
Epoch: 11 Training Loss:  1.680138225445662     Training Accuracy:  0.58427
Epoch: 12 Training Loss:  1.5025189302461532    Training Accuracy:  0.62545
Epoch: 13 Training Loss:  1.31784445954405 75   Training Accuracy:  0.67461
Epoch: 14 Training Loss:  1.1250148179281094    Training Accuracy:  0.72435
Epoch: 15 Training Loss:  0.9188690723665535    Training Accuracy:  0.78198
```

- Obtained the training loss and training accuracy plots for CrossEntropyLoss at 15 epochs.



- Then, saved the model's weights and biases using torch.save module and using the pickle library, we can import these trained weights and biases anytime again.
- Then, I visualized a certain class of the validation set.



- Then, tested the model on the validation data and obtained the following results.

```
Testing on Validation set Complete!
Top-5 Validation Accuracy:  0.375
Validation Loss:  3.0129431426525115    Test Accuracy:  0.3533203125
```

- Could not test the model on the test data because there are no labels provided for the test data in the dataset.
- Then, I defined the Triplet Loss class and used hard-mining to get the anchor, positive and negative samples.
- The triplet loss function aims to encourage the neural network to learn embeddings (vector representations) of input samples such that samples that are similar to each other are closer in the embedding space, while samples that are dissimilar are farther apart.
- The forward method of the TripletLoss class first computes the Euclidean distances between the anchor and positive samples (pos_dist) and between the anchor and negative samples (neg_dist). It then calculates the loss as the mean of the maximum of 0 and the difference between pos_dist and neg_dist, plus a margin value. The margin value is a hyperparameter that determines the minimum distance between the anchor and negative samples that the network should learn. If the distance between the anchor and negative samples is less than the margin, then the loss is 0, and the network does not receive any penalty for this sample triplet.
- The output of the function is a scalar tensor representing the triplet loss for the given anchor, positive, and negative samples. The objective of training a neural network with triplet loss is to minimize this loss function over a set of training samples.
- Started training the ResNet18 model with Triplet Loss for 10 epochs but it couldn't be trained due to GPU constraints and high computational costs. This problem arose because of the multiple layered architecture of ResNet18 model and due to the large 100k size of the dataset with image with pixel values 64*64.

- Then, Defined the Central Loss class.
- The center loss function aims to encourage the neural network to learn discriminative features by penalizing large intra-class variations of the learned features.

- The inputs to the function are a tensor x representing the features of the input samples and a tensor labels representing the ground truth class labels of the input samples. The function also takes as input some hyperparameters, including the number of classes (num_classes), the dimensionality of the feature space (feat_dim), and a regularization parameter (lambda_c) that controls the strength of the center loss penalty.
- The forward method of the CenterLoss class first computes the distances between the input features and the learned class centers. It then uses these distances to calculate the center loss, which is the sum of the Euclidean distances between the input features and their corresponding class centers, weighted by the number of samples in each class. The center loss is scaled by the regularization parameter lambda_c.

- Next, the forward method computes the softmax loss using the input features and the learned class centers. The softmax loss is a standard cross-entropy loss function that penalizes the difference between the predicted class probabilities and the ground truth class labels.
- Finally, the forward method returns the sum of the softmax loss and the scaled center loss, which is the total loss that is used to update the neural network weights during training.
- Note that the center loss function requires that the class centers be learned during training. In this implementation, the class centers are represented as a learnable parameter of the neural network and are initialized randomly. During training, the centers are updated using stochastic gradient descent along with the other model parameters.
- Started training the ResNet18 model with Central Loss for 10 epochs but it couldn't be trained due to GPU constraints and high computational costs. This problem arose because of the multiple layered architecture of ResNet18 model and due to the large 100k size of the dataset with image with pixel values 64*64.

END OF QUESTION 1.

Question 02

Aim: Implement a multi-layered classifier where weights of each layer is calculated greedily using layer-wise pretraining with the help of auto-encoders on STL-10 dataset. Train a classifier having X = [1024, 1000, 500, 256, 128, 64] (B20AI052) structure (excluding input and output layers) for classification task on the test set.

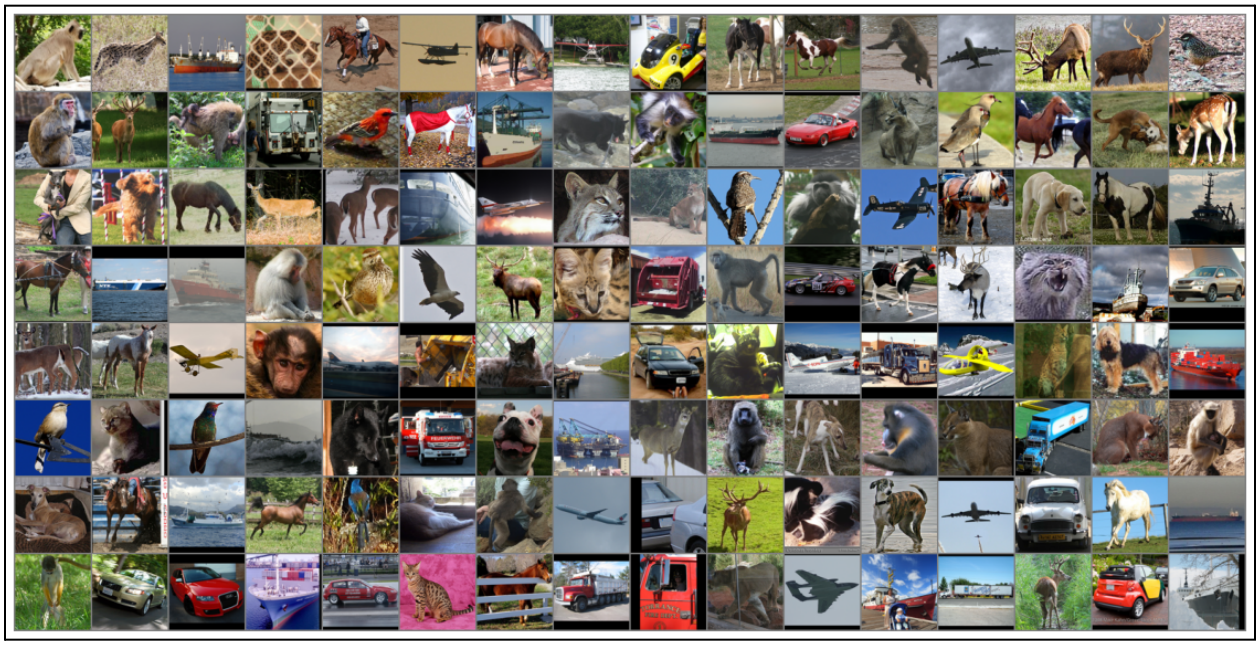
1. Report the classification accuracy on the test set and plot loss curves on the training and evaluation set.

2. Report the class-wise accuracy of each class.

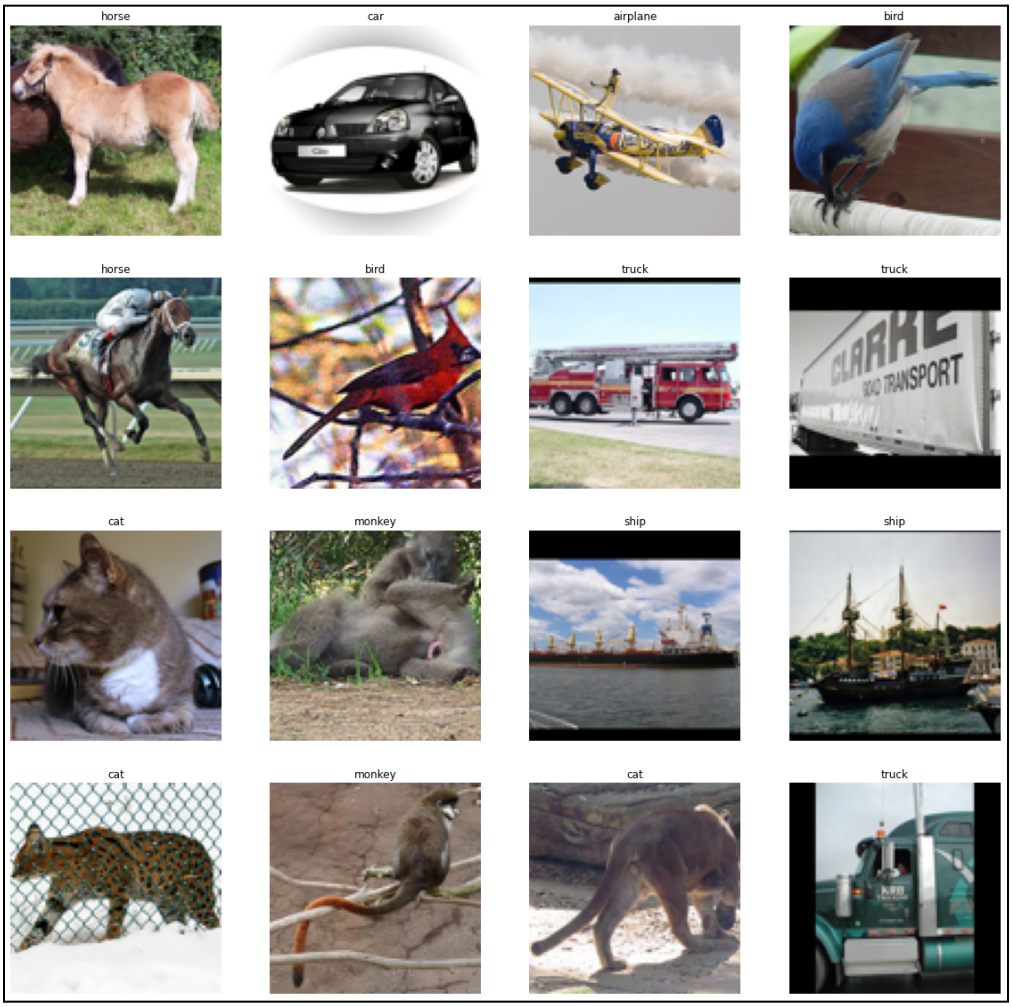
3. Plot t-sne for this model (use embeddings from layer X[3]) . Use the first 500 images of each class from the test dataset for this visualization.

Procedure:

- Imported all the necessary libraries.
- Then, loaded the STL10 dataset with batch_size = 128.
- Dataset has 5k training and 8k test samples with 10 classes = ('airplane', 'bird', 'car', 'cat', 'deer', 'dog', 'horse', 'monkey', 'ship', 'truck')
- Visualizing the dataset.



- Visualizing some images with their labels.



- Created an Autoencoder class with 1 encoding layer and 1 decoding layer using the ReLU activation function with parameters as size of the input layer and the size of the output layer.
- Size of one batch of images.

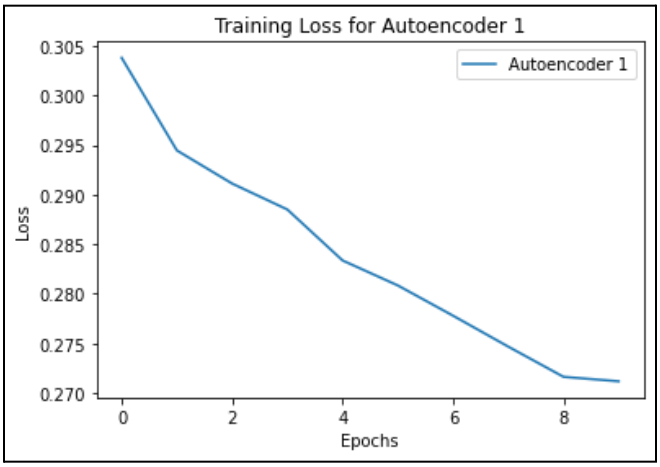
```
torch.Size([128, 3, 96, 96]) torch.Size([128])
```

- X = [1024, 1000, 500, 256, 128, 64]
- Then, created a list of 6 autoencoder objects.
- The input sizes of each AutoEncoder layer are defined in the list X, where the first element corresponds to the input size of the first AutoEncoder layer and the last

element corresponds to the output size of the last AutoEncoder layer. The code initializes a list of AutoEncoder objects, where each AutoEncoder object represents an individual layer of the deep autoencoder network.

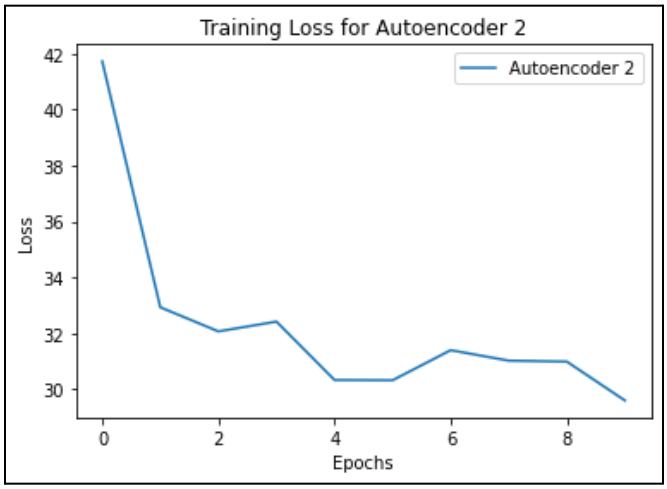
- There are six AutoEncoder objects created, with input and output sizes of [3x96x96, 1024], [1024, 1000], [1000, 500], [500, 256], [256, 128], and [128, 64].
- Once the pre-training is complete, the weights of the AutoEncoder layers can be used as the initial weights for a supervised learning task, such as classification.
- Trained the AutoEncoder1 for layer 1.

```
Epoch: 1 Training Loss:  0.3038179278373718
Epoch: 2 Training Loss:  0.29446936920285227
Epoch: 3 Training Loss:  0.29112856462597847
Epoch: 4 Training Loss:  0.28850892558693886
Epoch: 5 Training Loss:  0.2833679378032684
Epoch: 6 Training Loss:  0.280848516151309
Epoch: 7 Training Loss:  0.2777954451739788
Epoch: 8 Training Loss:  0.2746731698513031
Epoch: 9 Training Loss:  0.2716262027621269
Epoch: 10 Training Loss:  0.2711761757731438
Training complete!
```



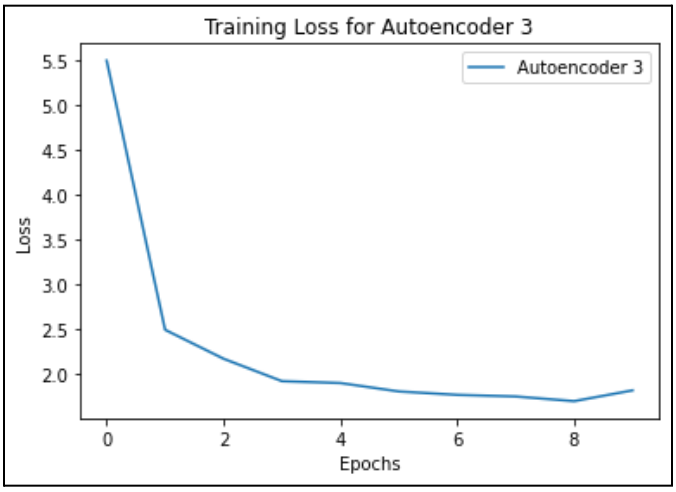
- Trained the AutoEncoder2 for layer 2.

```
Epoch: 1 Training Loss:  41.727115851640704
Epoch: 2 Training Loss:  32.93221936225891
Epoch: 3 Training Loss:  32.07056016921997
Epoch: 4 Training Loss:  32.42007880210876
Epoch: 5 Training Loss:  30.3319397732615 46
Epoch: 6 Training Loss:  30.3236543178558 34
Epoch: 7 Training Loss:  31.39849853515625
Epoch: 8 Training Loss:  31.02434377670288
Epoch: 9 Training Loss:  30.98940238952637
Epoch: 10 Training Loss:  29.59950248003006
Training complete!
```



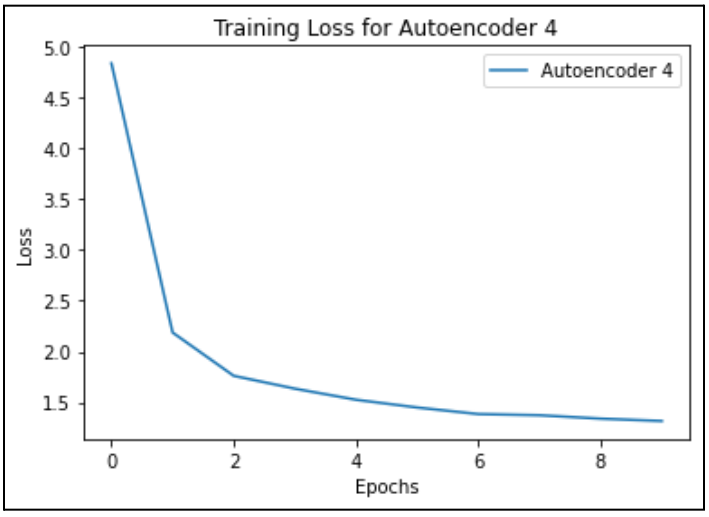
- Trained the AutoEncoder3 for layer 3.

```
Epoch: 1 Training Loss:  5.499083068966866
Epoch: 2 Training Loss:  2.4903332009911536
Epoch: 3 Training Loss:  2.1669549852609635
Epoch: 4 Training Loss:  1.9160076886415482
Epoch: 5 Training Loss:  1.8952501982450485
Epoch: 6 Training Loss:  1.8011401191353797
Epoch: 7 Training Loss:  1.7632052809000016
Epoch: 8 Training Loss:  1.7449606984853745
Epoch: 9 Training Loss:  1.6928534381091596
Epoch: 10 Training Loss:  1.813394846022129
Training complete!
```



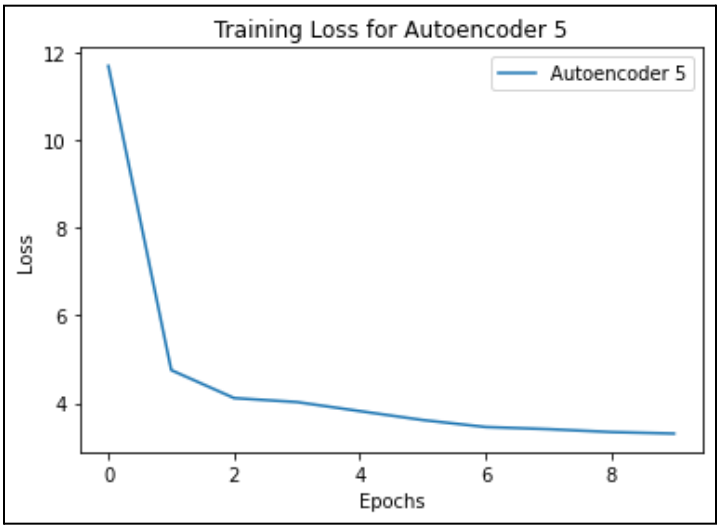
- Trained the AutoEncoder4 for layer 4.

```
Epoch: 1 Training Loss:  4.8335756182670595
Epoch: 2 Training Loss:  2.18724362552166
Epoch: 3 Training Loss:  1.7611338198184967
Epoch: 4 Training Loss:  1.6349599108099937
Epoch: 5 Training Loss:  1.5259217500686646
Epoch: 6 Training Loss:  1.449666327238083
Epoch: 7 Training Loss:  1.3851825997233391
Epoch: 8 Training Loss:  1.3736420139670371
Epoch: 9 Training Loss:  1.3401730954647064
Epoch: 10 Training Loss:  1.3171387895941735
Training complete!
```



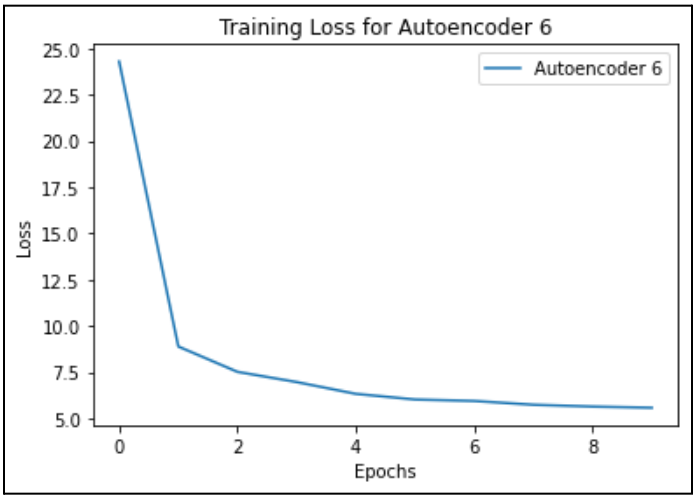
- Trained the AutoEncoder5 for layer 5.

```
Epoch: 1 Training Loss:  11.67928010225296
Epoch: 2 Training Loss:  4.754090788960457
Epoch: 3 Training Loss:  4.116962707042694
Epoch: 4 Training Loss:  4.026648128032685
Epoch: 5 Training Loss:  3.82041882276535
Epoch: 6 Training Loss:  3.617280888557434
Epoch: 7 Training Loss:  3.4585824638605116
Epoch: 8 Training Loss:  3.409708333015442
Epoch: 9 Training Loss:  3.3432934075593947
Epoch: 10 Training Loss:  3.308962282538414
Training complete!
```



- Trained the AutoEncoder6 for layer 6.

```
Epoch: 1 Training Loss:  24.299004673957825
Epoch: 2 Training Loss:  8.89013296365738
Epoch: 3 Training Loss:  7.518487584590912
Epoch: 4 Training Loss:  6.967295849323273
Epoch: 5 Training Loss:  6.328946959972382
Epoch: 6 Training Loss:  6.024802088737488
Epoch: 7 Training Loss:  5.944502758979797
Epoch: 8 Training Loss:  5.740398707985878
Epoch: 9 Training Loss:  5.642074960470199
Epoch: 10 Training Loss:  5.577557355165482
Training complete!
```



- Then, save the weights and biases of the all the 6 autoencoders using torch.save module.
- Then, created a Classifier class with its layers as the embeddings obtained from the six different autoencoders.

```
Classifier(
  (dropout): Dropout(p=0.2, inplace=False)
  (final): Sequential(
    (0): Linear(in_features=27648, out_features=1024, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1024, out_features=1000, bias=True)
    (3): ReLU()
    (4): Linear(in_features=1000, out_features=500, bias=True)
    (5): ReLU()
    (6): Linear(in_features=500, out_features=256, bias=True)
    (7): ReLU()
    (8): Linear(in_features=256, out_features=128, bias=True)
    (9): ReLU()
    (10): Linear(in_features=128, out_features=64, bias=True)
    (11): ReLU()
    (12): Linear(in_features=64, out_features=10, bias=True)
    (13): LogSoftmax(dim=1)
  )
)
```

- Using the Adam optimizer with learning rate=0.001 and NLLLoss.
- Then, trained the classifier for 20 epochs and obtained the following results.

```
Epoch: 1 Training Loss:  2.657470792531967 Training Accuracy:  0.1568
Epoch: 2 Training Loss:  1.924582263827324 Training Accuracy:  0.2668
Epoch: 3 Training Loss:  1.8103835791349412 Training Accuracy:  0.2966
Epoch: 4 Training Loss:  1.7165536642074586 Training Accuracy:  0.3338
Epoch: 5 Training Loss:  1.606030285358429 Training Accuracy:  0.374
Epoch: 6 Training Loss:  1.523705893754959 Training Accuracy:  0.4088
Epoch: 7 Training Loss:  1.421444556117058 Training Accuracy:  0.462
Epoch: 8 Training Loss:  1.3754271477460862 Training Accuracy:  0.482
Epoch: 9 Training Loss:  1.2950413256883622 Training Accuracy:  0.5302
Epoch: 10 Training Loss:  1.1330712661147118 Training Accuracy:  0.5732
Epoch: 11 Training Loss:  1.019545416533947 Training Accuracy:  0.6194
Epoch: 12 Training Loss:  0.917233356833458 Training Accuracy:  0.6712
Epoch: 13 Training Loss:  0.8722163900732994 Training Accuracy:  0.6782
Epoch: 14 Training Loss:  0.8396431557834149 Training Accuracy:  0.6984
Epoch: 15 Training Loss:  0.730830492079258 Training Accuracy:  0.7352
Epoch: 16 Training Loss:  0.734868835657835 Training Accuracy:  0.7368
Epoch: 17 Training Loss:  0.6171531245112419 Training Accuracy:  0.7796
Epoch: 18 Training Loss:  0.616012980043888 Training Accuracy:  0.782
Epoch: 19 Training Loss:  0.5562051787972451 Training Accuracy:  0.8098
Epoch: 20 Training Loss:  0.5033839508891106 Training Accuracy:  0.8208
Training complete!
```

- Then, tested the classifier on the testing data and obtained the following results.

```
Accuracy of the network on the 10000 test images: 38.4 %
```

- Also, obtained the class-wise accuracy of the classifier.

```
Accuracy of airplane : 70 %
Accuracy of  bird : 45 %
Accuracy of   car : 42 %
Accuracy of   cat : 10 %
Accuracy of  deer : 55 %
Accuracy of   dog : 25 %
Accuracy of horse : 33 %
Accuracy of monkey : 31 %
Accuracy of  ship : 52 %
Accuracy of truck : 40 %
```

- Then, plotted t-SNE for this model using embedding from X[3] from the autoencoder and using the first 500 images of each class from the test dataset for this visualization.

- Initialized empty lists **lt_embed** and **lt_lb** to store the encoded representations and labels of the images. The variable **ln** is set to 0 to keep track of the number of processed images.
- Then used a **torch.no_grad()** context manager to disable gradient computations for faster inference. It iterates over the images and labels in the testloader_STL10 data loader, which is assumed to contain image-label pairs.
- For each image, the code passes it through the AutoEncoder layers in sequence to obtain its encoded representation. The output of each AutoEncoder layer is passed as input to the next AutoEncoder layer, until the final encoded representation is obtained from the last AutoEncoder layer.
- The encoded representation tensor of each image is appended to the lt_embed list, while the corresponding label is appended to the lt_lb list. The variable ln is updated with the number of processed images, and the loop exits when ln is greater than or equal to 500.
- Finally, the encoded representation tensor and label tensor are concatenated along the first dimension using torch.cat() to obtain the encoded representations of all the processed images and their corresponding labels, respectively.
- t-SNE Plot.