EO270: Assignment-1

LOCHAN SURYA TEJA NEELI(06-18-01-10-22-23-1-23198)

October 29, 2024

1 Softmax Regression

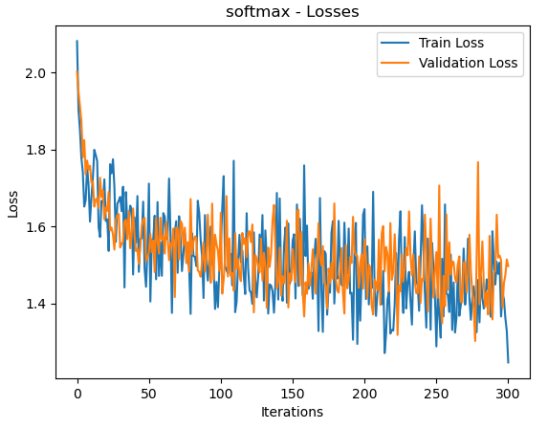
1.1 Logistic Regression

Early stopped when the validation accuracy became greater than 80% and obtained 81.5%(approximate) validation accuracy.

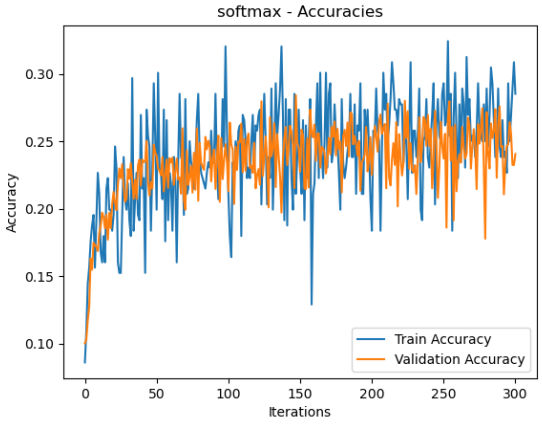
1.2 Experimentation

Performed several experiments by varying the batch sizes but the

1.3 Multi-class Logistic Regression



(a) Batch Size=256(default);Softmax Loss for 3000 iterations with no gradient clipping



(b) Batch Size=256(default);Softmax accuracy for 3000 iterations with no gradient clipping

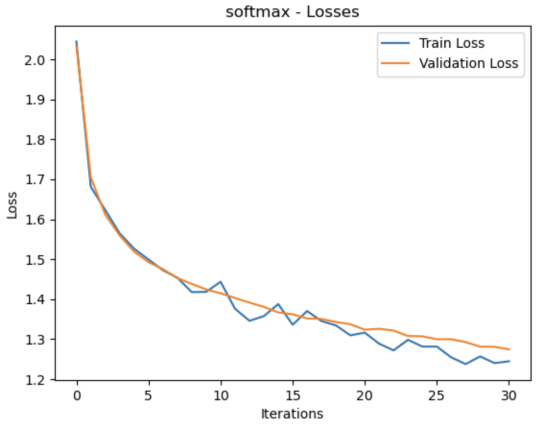


Figure 2: Batch size: 4096, num_iters: 3000 with gradient clipping

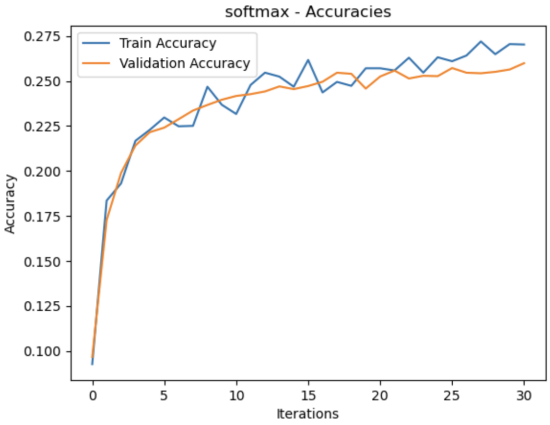


Figure 3: Batch size: 4096, num_iters: 3000 with gradient clipping

1.3.1 Observation:

The graphs became more understandable when the gradient clipping is implemented rather than the normal implementation without it.

The validation accuracy has plateaued at 26%. This outcome was anticipated owing to the loss of spatial information inherent in flattening the image tensor. Consequently, implementing a Convolutional Neural Network (CNN) architecture is advisable. CNNs leverage weight-sharing or parameter sharing, resulting in fewer parameters to update during inference, when compared to a fc-layer implementation particularly when evaluating the model on the validation dataset.

By transitioning to a CNN architecture, we can preserve spatial information more effectively, enhancing the model's ability to learn intricate patterns within the data. This approach aligns with the inherent strengths of CNNs, which excel at processing grid-like data such as images. With parameter sharing, the model becomes more efficient in capturing and generalizing features across different regions of the input image, leading to improved performance metrics such as validation accuracy.

1.3.2 Binary Cross Entropy and Cross Entropy

- The `calculate_loss` function calculates the loss of a linear model on a given dataset. It takes as input the model (`LinearModel`), the features (`X`), and the labels (`y`). The `is_binary` parameter indicates whether the task is binary classification or not. The function returns the loss value, which is computed using the following formula:

$$\text{loss} = \begin{cases} -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i + \epsilon) + (1 - y_i)\log(1 - \hat{y}_i + \epsilon)\right) & \text{if binary classification} \\ -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij} + \epsilon) & \text{if multi-class classification} \end{cases}$$

  where N is the number of samples, y_i is the true label of the i-th sample, $\hat{y}_i$ is the predicted probability of the i-th sample belonging to the positive class, y_{ij} is the indicator function for the i-th sample being in class j, $\hat{y}_{ij}$ is the predicted probability of the i-th sample belonging to class j, and ϵ is a small constant to avoid numerical instability(to avoid $\log(0)$).

- The `calculate_accuracy` function calculates the accuracy of a linear model on a given dataset. It also takes the model (`LinearModel`), features (`X`), and labels (`y`) as input. The `is_binary` parameter indicates whether the task is binary classification or not. The accuracy is computed using the following formula:

$$\text{accuracy} = \begin{cases} \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(\hat{y}_i > 0.5) & \text{if binary classification} \\ \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(\text{argmax}(\hat{y}_i) = y_i) & \text{if multi-class classification} \end{cases}$$

  where N is the number of samples, $\hat{y}_i$ is the predicted probability (or class label) of the i-th sample, y_i is the true label of the i-th sample, and $\mathbb{1}(\cdot)$ is the indicator function.

Hyperparameter	**Value**
batch_size	256
num_iters	3000
learning rate	0.001
gradient clipping	implemented
regularization (l_2_lambda)	0.1
early stopping	implemented when validation accuracy reached above 30%
patience	0

Table 1: Soft-max Regression Model Hyperparameters

1.3.3 Early Stopping for Softmax Regression

Algorithm 1: Early Stopping in Machine Learning Training

Input: Training and validation datasets, model, hyperparameters
Output: Trained model with early stopping

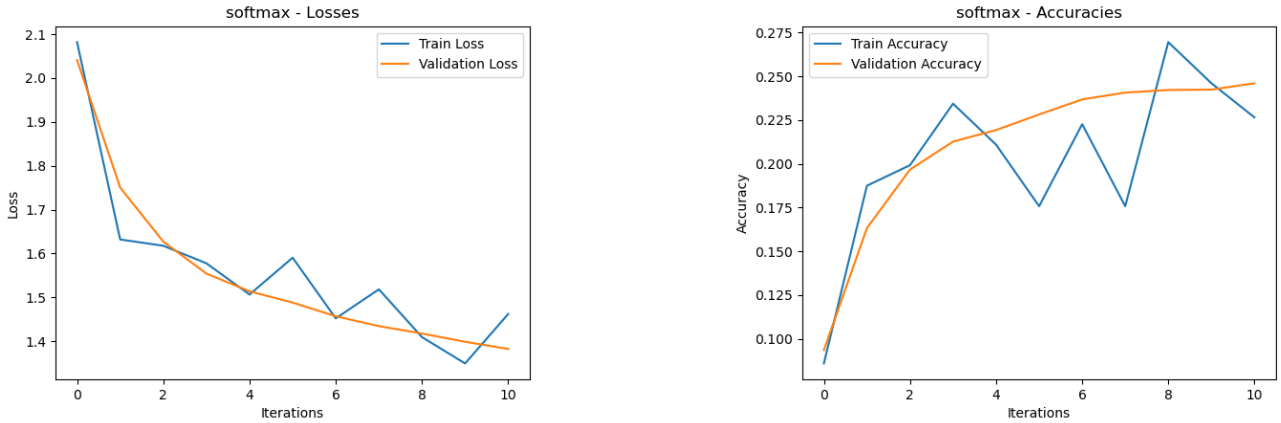
```
Initialize best validation loss/accuracy and counter;
for epoch ← 1 to num_epochs do
    Train the model on the training dataset;
    Evaluate the model on the validation dataset;
    if validation metric improves then
        Update best validation loss/accuracy;
        Reset counter;
    end
    else
        Increment counter;
    end
    if counter ≥ patience then
        Terminate training (early stopping);
    end
end
Report final training and validation metrics;
```

Here, the **patience** is a parameter for implementing the early stopping, representing how many epochs to wait for improvement before early stopping.

The validation metric can be the loss or the accuracy.

Hyperparameter	Value
batch_size	256
lr	1e-4
l2_lambda	0.2
grad_norm_clip	5.0
patience	50

Table 2: Hyperparameters for softmax



(a) Softmax loss vs iterations

(b) Accuracy vs iterations

Figure 4: Comparison of softmax loss and accuracy vs iterations for patience

2 Contrastive Representation Learning

2.1 Contrastive Representation

(i)d_{out} : 64
(ii)**Architecture :** VGG-16 along with batch normalization
(iii)**Activation:**ReLU

The following is the torch summary of the architecture used for the Encoder class:

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
Sequential: 1-1                          [-1, 512, 1, 1]           --
|    Conv2d: 2-1                         [-1, 64, 32, 32]          1,792
|    BatchNorm2d: 2-2                    [-1, 64, 32, 32]          128
|    ReLU: 2-3                           [-1, 64, 32, 32]          --
|    Conv2d: 2-4                         [-1, 64, 32, 32]          36,928
|    BatchNorm2d: 2-5                    [-1, 64, 32, 32]          128
|    ReLU: 2-6                           [-1, 64, 32, 32]          --
|    MaxPool2d: 2-7                      [-1, 64, 16, 16]          --
|    Conv2d: 2-8                         [-1, 128, 16, 16]         73,856
|    BatchNorm2d: 2-9                    [-1, 128, 16, 16]         256
|    ReLU: 2-10                          [-1, 128, 16, 16]         --
|    Conv2d: 2-11                        [-1, 128, 16, 16]         147,584
|    BatchNorm2d: 2-12                   [-1, 128, 16, 16]         256
|    ReLU: 2-13                          [-1, 128, 16, 16]         --
|    MaxPool2d: 2-14                     [-1, 128, 8, 8]           --
|    Conv2d: 2-15                        [-1, 256, 8, 8]           295,168
|    BatchNorm2d: 2-16                   [-1, 256, 8, 8]           512
|    ReLU: 2-17                          [-1, 256, 8, 8]           --
|    Conv2d: 2-18                        [-1, 256, 8, 8]           590,080
|    BatchNorm2d: 2-19                   [-1, 256, 8, 8]           512
|    ReLU: 2-20                          [-1, 256, 8, 8]           --
|    Conv2d: 2-21                        [-1, 256, 8, 8]           590,080
|    BatchNorm2d: 2-22                   [-1, 256, 8, 8]           512
|    ReLU: 2-23                          [-1, 256, 8, 8]           --
|    MaxPool2d: 2-24                     [-1, 256, 4, 4]           --
|    Conv2d: 2-25                        [-1, 512, 4, 4]           1,180,160
|    BatchNorm2d: 2-26                   [-1, 512, 4, 4]           1,024
|    ReLU: 2-27                          [-1, 512, 4, 4]           --
|    Conv2d: 2-28                        [-1, 512, 4, 4]           2,359,808
|    BatchNorm2d: 2-29                   [-1, 512, 4, 4]           1,024
|    ReLU: 2-30                          [-1, 512, 4, 4]           --
|    Conv2d: 2-31                        [-1, 512, 4, 4]           2,359,808
|    BatchNorm2d: 2-32                   [-1, 512, 4, 4]           1,024
|    ReLU: 2-33                          [-1, 512, 4, 4]           --
|    MaxPool2d: 2-34                     [-1, 512, 2, 2]           --
|    Conv2d: 2-35                        [-1, 512, 2, 2]           2,359,808
|    BatchNorm2d: 2-36                   [-1, 512, 2, 2]           1,024
|    ReLU: 2-37                          [-1, 512, 2, 2]           --
|    Conv2d: 2-38                        [-1, 512, 2, 2]           2,359,808
|    BatchNorm2d: 2-39                   [-1, 512, 2, 2]           1,024
|    ReLU: 2-40                          [-1, 512, 2, 2]           --
```

```
|    Conv2d: 2-41                        [-1, 512, 2, 2]           2,359,808
|    BatchNorm2d: 2-42                   [-1, 512, 2, 2]           1,024
|    ReLU: 2-43                          [-1, 512, 2, 2]           --
|    MaxPool2d: 2-44                     [-1, 512, 1, 1]           --
AdaptiveAvgPool2d: 1-2                   [-1, 512, 7, 7]           --
Linear: 1-3                              [-1, 64]                  1,605,696
==========================================================================================
Total params: 16,328,832
Trainable params: 16,328,832
Non-trainable params: 0
Total mult-adds (M): 329.52
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 4.22
Params size (MB): 62.29
Estimated Total Size (MB): 66.52
==========================================================================================
```

2.2 Training Strategy

2.2.1 Cost Function

Triplet Margin Loss: The Triplet Margin Loss function is utilized to train the model. It is formulated as follows:

$$L(X_a, X_p, X_n) = \max\{d(X_a, X_p) - d(X_a, X_n) + \text{margin}, 0\}$$
$$d(x, y) = ||x - y||_p^2$$

Where:

- X_a: Anchor Sample
- X_p: Positive Sample
- X_n: Negative Sample
- margin: A positive value (set to 1.0 in implementation)

The Triplet Margin Loss compares the distance between the anchor sample (X_a) and the positive sample (X_p) with the distance between the anchor sample and the negative sample (X_n). The loss encourages the model to minimize the distance between the anchor and positive samples while maximizing the distance between the anchor and negative samples, with a margin to ensure a gap between them.

We employ the Euclidean distance (L_2 norm) to compute the distance between samples.

This loss function is readily available in PyTorch, simplifying its integration into the model training and evaluation pipeline.

During training, the loss is sampled batch-wise for plotting every 10 iterations. While fluctuations may occur, they are generally not significant and do not substantially impact the training process.

2.2.2 Batch Size

The batch size chosen was 2048. The reason for this design choice is to reduce the training time of the model and optimally use the GPU(Nvidia Quadro GV100) memory, and also using higher batch sizes will sample the data well for the positive and negative samples for a given anchor sample for the loss evaluation.

2.2.3 Image Sampling for Contrastive Learning

The images are sampled to support contrastive learning, where the model learns to distinguish between similar (from the same class) and dissimilar (from different classes) images.

1. **Shuffling:** The dataset is shuffled to ensure random batches without any inherent order from the original dataset.
2. **Batching:** Batches of images are generated from the shuffled dataset, with the batch size determined by the parameter `batch_size`.
3. **Anchor, Positive, and Negative Sampling:** For each image in the batch:
   - **Anchor sample:** The current image in the batch.
   - **Positive sample:** Randomly selected from images in the batch with the same class label as the anchor.
   - **Negative sample:** Randomly selected from images in the batch with different class labels from the anchor.

   This sampling is performed using numpy functions such as `np.where` to find indices and `np.random.choice` for random selection.

This process results in three arrays(images): X_a (anchor samples), X_p (positive samples), and X_n (negative samples), which are then used for training the model.

2.2.4 Criteria for Stopping the training:

Early stopping is implemented in training loop when the model saturated(no significant difference between the successive loss values and successive accuracy values for the validation dataset) for the validation accuracy. But, for the experimental results, there was no stopping criteria implemented,to increase the validation accuracy.

Also, since the training(epochs) is not done exactly for whole dataset, since we were sampling randomly across the whole dataset. So, to make the model see the whole training dataset in expectation, and also the model being deep, I trained it for 2000 iterations.

Hyperparameter	value
criteria	loss
patience	300
num_iters	2000

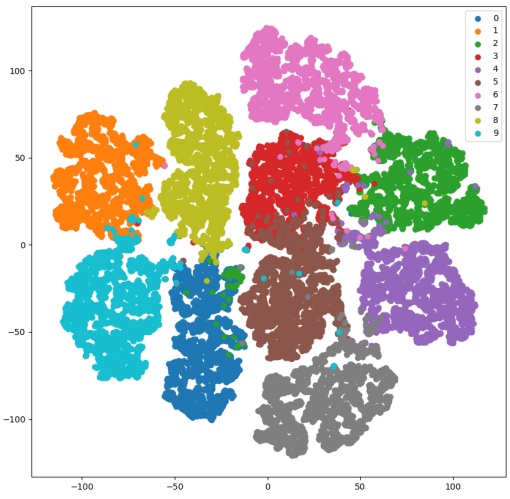
Table 3: Early Stopping for cont_rep

Algorithm 2: Fit_Model

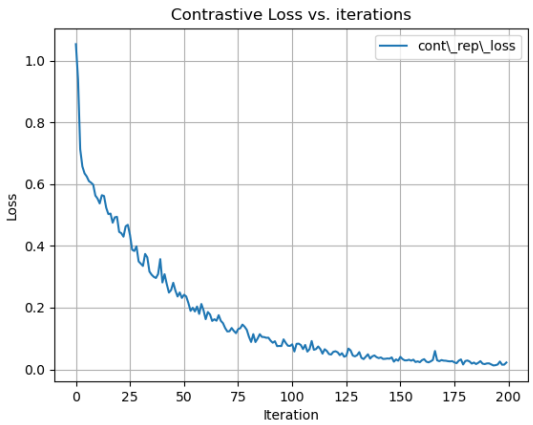
Input: encoder, classifier, X_train, y_train, X_val, y_val, args
Output: train_losses, train_accs, val_losses, val_accs

```
Initialize empty lists: train_losses, train_accs, val_losses, val_accs;
if args.mode is 'fine_tune_linear' then
    Extract embeddings from encoder for X_train and X_val;
    Fit a linear model to the embeddings using X_train, y_train, X_val, y_val;
    Store the resulting train and val losses and accuracies;
end
else
    Initialize Adam optimizer and cross-entropy loss function;
    Set encoder and classifier to training mode;
    for i from 1 to args.num_iters do
        Sample a batch from X_train and y_train;
        Forward pass: compute predictions using encoder and classifier;
        Compute loss;
        Backward pass: compute gradients and update model parameters;
        if i % 10 == 0 then
            Store current train loss and accuracy;
            Evaluate model on validation set;
            Store validation loss and accuracy;
            Print current iteration and performance metrics;
        end
    end
end
return train_losses, train_accs, val_losses, val_accs;
```

2.3 t-Stochastic Neighbors Embedding(t-SNE)



(a) z_dim= 64; t-SNE plot with margin=1 and iterations 2000; y-axis: t-SNE dimension 2, x-axis: t-SNE dimension 1; batch_size=2048



(b) Contrastive representation loss vs epoch;(please refer to the file "Cont_rep_loss.ipynb" in the zip folder for this plot)

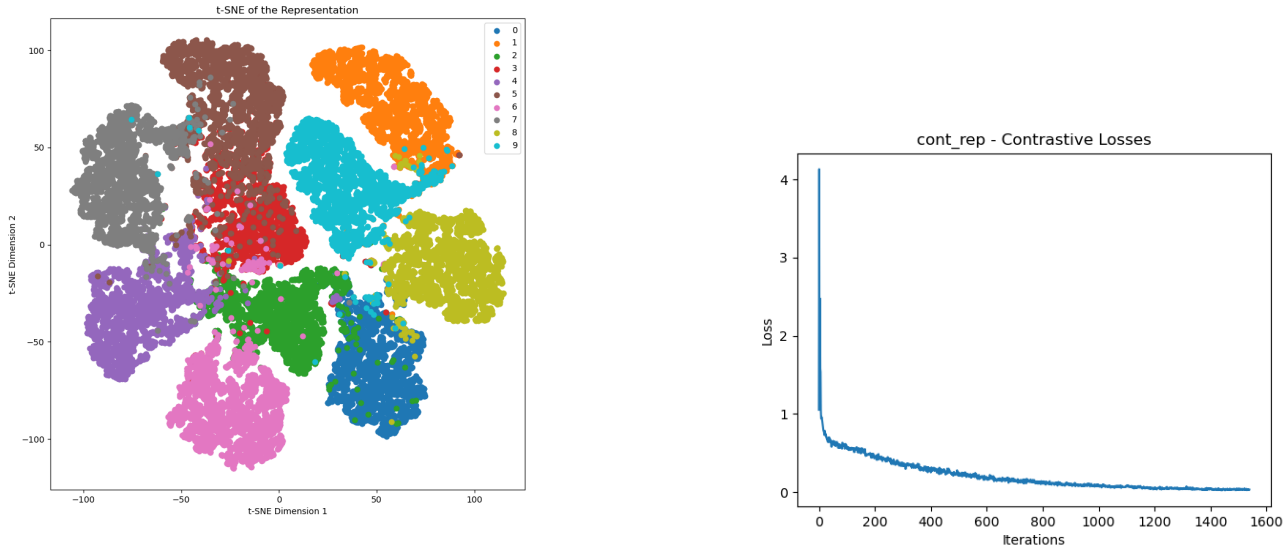
Figure 5: t-SNE plot and contrastive representation loss(z_dim=64; batch_size=2048)

Plot is saved as tsne-2000.png in images folder

2.3.1 Inference from the t-SNE plot

The graph clearly shows that the clusters are distinctly separated. This suggests that images belonging to the same class are grouped closely together, while images from different classes are well separated. It indicates that our encoder architecture effectively encoded the images, and the model learned effectively. This gives us confidence to proceed with fine-tuning the model, as neural networks are known to excel at capturing complex relationships between inputs and outputs.

2.3.2 Early Stopping



(a) tsne plot for z_dim=32 with early stopping

(b) Cost Function (loss function) vs iterations

Figure 6: Early Stopping implementation for contrastive learning with patience=50; early stopped at iteration 1560; batch size=2048; margin=1.0

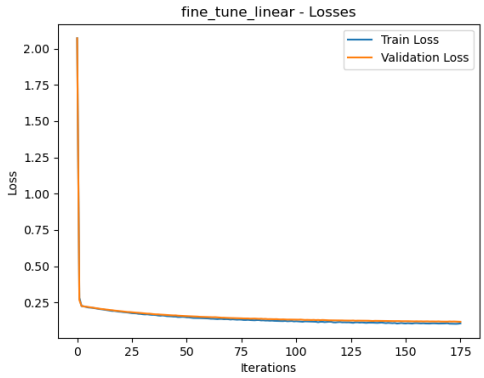
The leaderboard score was obtained when z_dim=64 and from the figure you can say that z_dim=64 t-SNE plot is better since the label 3 is well separated from the other labels in the plot.

2.4 Fine Tuning Linear Classifier Using the Learned Vectors

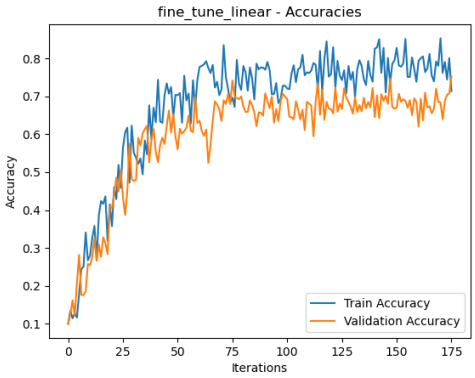
Fine_tune_linear did better than the trivial soft-max regression where we implemented:

Hyperparameter	value
z_dim	32
criteria	loss
patience	300
num_iters	4000
early stopped at iteration	2900
train_accuracy	89.99%
train_loss	0.1124
validation_accuracy	77.44%
validation_loss	0.1257

Table 4: Hyperparameters and metrics



(a) fine_tune_linear - loss vs iterations



(b) fine_tune_linear - accuracy vs iterations

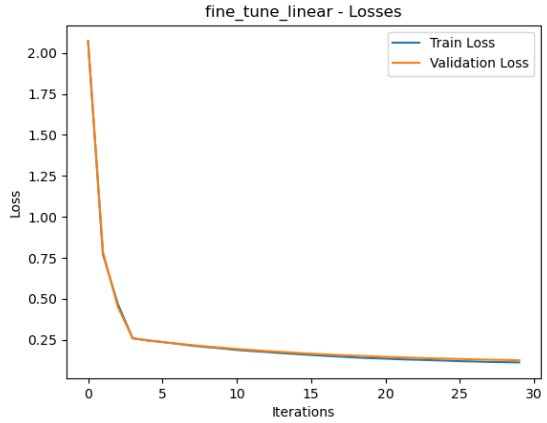
Figure 7: fine tuning: loss and accuracy for fine_tune_linear; z_dim=64

As the model saturated at this number of iterations, when ran for 50000 iterations, in the later experiment, I early stopped it at the point where the validation accuracy hit 75%.

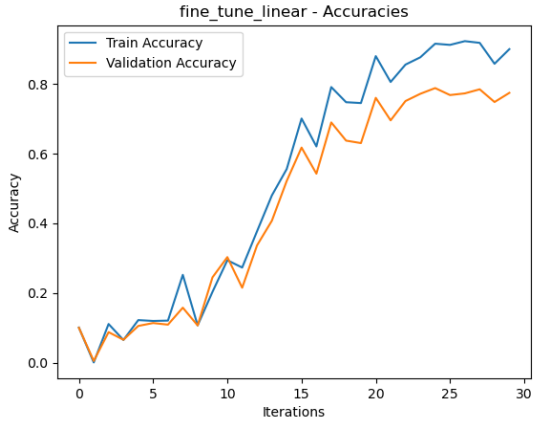
2.4.1 Early stopping

Hyperparameter	Value
z_dim	32
mode	fine_tune_linear
batch_size	2048
patience	300
lr	0.0001
num_iters	4000
early stopped at iteration	-

Table 5: Hyperparameters for fine-tune-linear mode



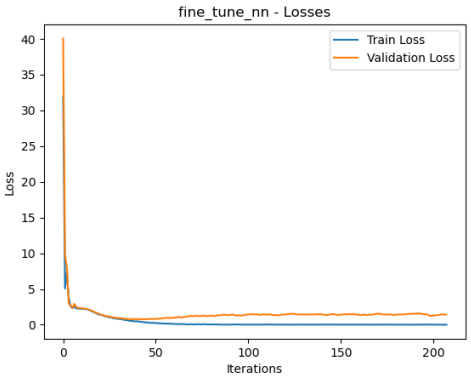
(a) Fine tune linear Loss vs iterations with early stopping



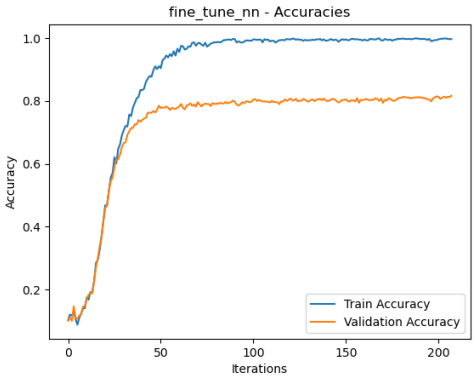
(b) Fine tune linear Accuracy vs iterations with early stopping

Figure 8: Fine tune linear Loss and Accuracy vs iterations with early stopping

2.5 Fine Tuning Classifier Layer



(a) fine_tune_nn-loss vs iterations; for 2500 iterations



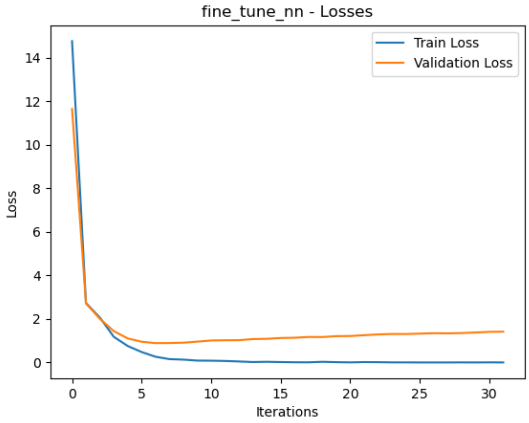
(b) fine_tune_nn-accuracy vs iterations; for 2500 iterations

Figure 9: fine_tune_nn: Loss and accuracy; z_dim=64

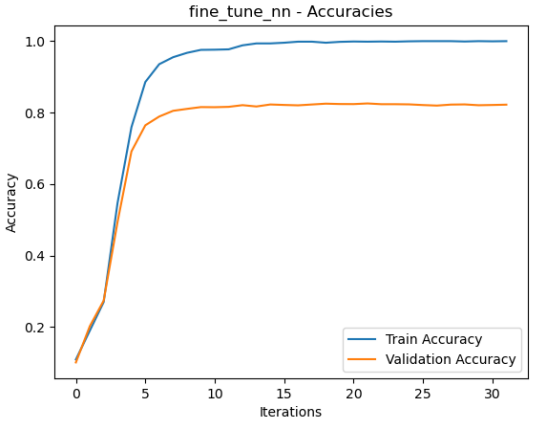
2.5.1 Early Stopping

Hyperparameter	Value
z_dim	32
Batch Size	2048
Patience	10
Learning Rate	0.0001
Number of Iterations	4000
Early Stopped at iteration	310

Table 6: Hyperparameters for fine_tune_nn



(a) Loss vs iterations with early stopping



(b) Accuracy vs iterations with early stopping

Figure 10: Fine tune NN Loss and Accuracy vs iterations with early stopping

Metric	Value
Training Loss	0.0184
Training Accuracy	99.85%
Validation Loss	1.2547
Validation Accuracy	82.57%

Table 7: Model Metrics

- For 1000 iterations, fine_tune_linear validation accuracy was almost 70%, whereas for the same number of iterations, it was 80%.

After fine-tuning the neural network classifier utilizing the embeddings extracted from the encoder, superior results were achieved compared to the softmax regression approach. The enhanced performance can be attributed to several factors:

1. **Minimization of Reconstruction Error**: The linear layer in the neural network tries to minimize the reconstruction error. This is achieved by optimizing the parameters of the linear layer to better reconstruct the original input.

2. **Utilization of Adam Optimizer**: The Adam optimizer is employed to update both the parameters of the linear layer and the encoder. This adaptive optimization algorithm enhances the convergence of the training process, leading to improved performance.

3. **Preservation of Spatial Information**: The softmax regression model, by contrast, was trained directly on the original images without utilizing embeddings. Consequently, when employing a flattened image tensor as input to the softmax layer, a significant amount of spatial information is lost during training. This loss of spatial information hampers the model's ability to capture intricate patterns and features within the data.

4. **Limited Capacity of Softmax Regression**: Given the inherent limitations of softmax regression, further performance gains are constrained when attempting to solely fine-tune this model. The training on flattened image tensor fed into the softmax regression layer fails to effectively leverage the richer feature representations extracted by the encoder.

2.6 Modifying Linear Classifier by Adding more than 1 fc-layer

Here is an excerpt from the terminal output for fine_tune_nn(z_dim=64, batch_size=2048, **classifier with one fc layer**):

```
Iter: 1850, Train Loss: 0.008415071293711662, Train Acc: 0.998046875,
Val Loss: 1.3419103622436523, Val Acc: 0.830299973487854
Iter: 2100, Train Loss: 0.0024710725992918015, Train Acc: 0.99951171875,
Val Loss: 1.3157786130905151, Val Acc: 0.8301999568939209

overfit:
Iter: 14990, Train Loss: 1.9557747421572458e-08, Train Acc: 1.0,
Val Loss: 1.6543446779251099, Val Acc: 0.863099992275238
```

The above mentioned metrics are for z_dim=32 was run after modifying the linear classifier layer to the following:

Layer (type)	Input Shape	Output Shape	Param #
Linear-1	[64, 512]	[64, 512]	262,656
ReLU-2	[64, 512]	[64, 512]	0
Linear-3	[64, 512]	[64, 10]	5,130

Table 8: Model Summary

The longer training time for fine_tune_nn compared to fine_tune_linear can be attributed to the additional nn classifier layer in the architecture employed in the neural network model.

Best Test Accuracy:86.35% for z_dim= 64; batch_size=2048

Plots are saved as:

```
plt.savefig(f'images/acc-{title}.png')
plt.savefig(f'images/loss-{title}.png')
```