Instructions to Run

Folder Structure

```
.
└── Contrastive_Representation
    ├── Report.pdf
    ├── ContrastiveRepresentation
    │   ├── __init__.py
    │   ├── main.py
    │   ├── model.py
    │   ├── pytorch_utils.py
    │   └── train_utils.py
    ├── images
    │   ├── 1.1a.png
    │   ├── 1.1b.png
    │   ├── 1.3.png
    │   ├── 2.2.png
    │   ├── acc.png
    │   └── loss.png
    ├── LogisticRegression
    │   ├── __init__.py
    │   ├── main.py
    │   ├── model.py
    │   └── train_utils.py
    ├── models
    │   └── encoder30.pth
    ├── run.py
    ├── utils.py
    └── output.txt
```

Listing 1: **Command to run project**

```
python run.py \

--mode name_of_mode \
<other hyper-params here>
```

Please check out output.txt to see iterative outputs obtained while fine tuning

Introduction

This assignment, designated as E0 270: Assignment 1, has a comprehensive exploration of two key topics: Softmax Regression and Contrastive Representation Learning.

Softmax Regression, also known as multi-class logistic regression, forms the base of many classification tasks in machine learning. It involves extending binary logistic regression to handle multiple classes, making it an one of the best tools for multi-class classification problems. In this assignment, we are tasked with implementing Softmax Regression using PyTorch, a popular deep learning framework, to classify images from the CIFAR-10 dataset.

Contrastive Representation Learning, on the other hand, is part of unsupervised learning, specifically focusing on learning representations of data in a way that enhances their discriminatory power. By training a neural network to maximize the similarity between similar instances while minimizing the similarity between dissimilar instances, Contrastive Representation Learning aims to create embeddings that capture meaningful relationships within the data. This assignment requires students to build a neural network architecture and train it using a contrastive loss function.

This paper will cover basic concepts of Softmax Regression and Contrastive Representation Learning, implementation of problem statement.

Data Description

The dataset used for this assignment is the CIFAR-10 dataset, which can be accessed at `https://www.cs.toronto.edu/kriz/cifar.html`. CIFAR-10 is a widely used benchmark dataset in the field of computer vision and machine learning, consisting of 60,000 color images divided into 10 classes. Each image in the dataset is a 3-channel RGB image with dimensions of 32×32 pixels.

The dataset is split into 50,000 training images and 10,000 test images, with each class containing an equal number of images. This balance ensures that the dataset is suitable for evaluating classification algorithms across different classes without bias.

In the context of this assignment, each data point in the CIFAR-10 dataset represents a $3 \times 32 \times 32$ image. When considering a linear model such as multi-class logistic regression, the features of each data point are the individual pixel values of the image. On the other hand, when utilizing convolutional neural networks (CNNs), the RGB channels of the image serve as the input features for the model.

The CIFAR-10 dataset provides an excellent platform for training and evaluating machine learning models for image classification tasks. By leveraging this dataset, students have the opportunity to explore various techniques and algorithms in the context of real-world image data, gaining valuable insights into the principles of computer vision and deep learning.

April 19, 2024

1 Problem 1: Softmax Regression

1.1 Softmax Regression

Softmax regression, also known as multinomial logistic regression, is a classification algorithm that generalizes logistic regression to multiple classes. It is commonly used in machine learning for multi-class classification tasks.

In softmax regression, we model the probability distribution over K different classes using a softmax function. Given an input vector $\mathbf{x}$, the softmax function computes the probability that $\mathbf{x}$ belongs to each class k as follows:

$$P(y = k|\mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{e^{\mathbf{w}_k^T \mathbf{x} + b_k}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T \mathbf{x} + b_j}} \tag{1}$$

where:

- $P(y = k|\mathbf{x}; \mathbf{W}, \mathbf{b})$ is the probability that $\mathbf{x}$ belongs to class k,
- $\mathbf{W}$ is the weight matrix,
- $\mathbf{b}$ is the bias vector,
- $\mathbf{w}_k$ is the k-th row of $\mathbf{W}$, and
- b_k is the k-th element of $\mathbf{b}$.

The softmax function ensures that the predicted probabilities sum up to 1 across all classes, making it suitable for multi-class classification. During training, the model learns the parameters $\mathbf{W}$ and $\mathbf{b}$ by minimizing a suitable loss function, such as the cross-entropy loss.

Softmax regression is commonly used in neural networks as the output layer for multi-class classification tasks. It forms an essential component of many deep learning architectures and is widely used in applications such as image classification, natural language processing, and sentiment analysis.

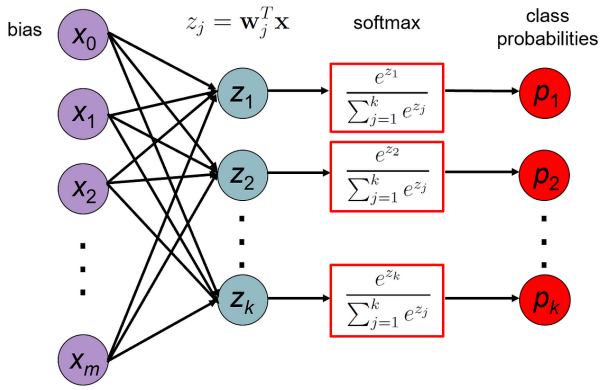


Figure 1: Softmax Regression

```
exp_values = np.exp(x - np.max(x, axis=-1, keepdims=True))
# exps = np.exp(shift_x)
softmax_values = exp_values / np.sum(exp_values, axis=-1, keepdims=True)
return softmax_values
```

1.2 Implementation

The `LinearModel` class serves as the foundation for our implementation. It initializes with random weights and a bias term, allowing it to represent a basic linear model. The `forward` method is defined to compute the output of the linear model given an input vector.

Logistic Regression: The `LogisticRegression` class extends the `LinearModel` class and specializes in logistic regression. Logistic regression is a binary classification algorithm that models the probability of a binary outcome. In our implementation, we use the sigmoid function to compute probabilities, and we override the `forward` method to apply the sigmoid function to the output of the linear model.

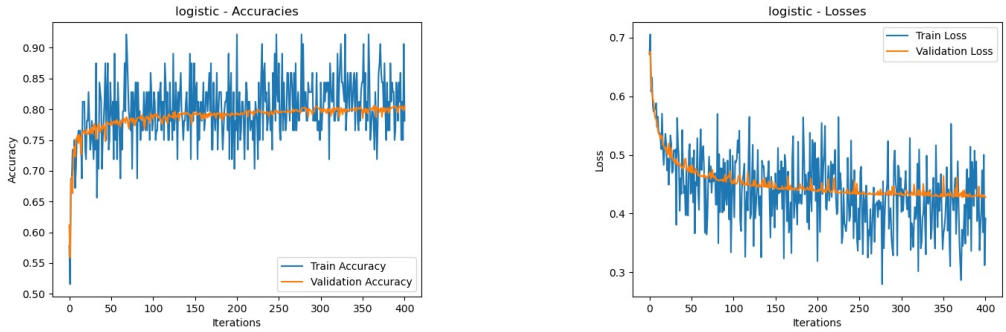
Softmax Regression The `SoftmaxRegression` class is another extension of the `LinearModel` class, tailored for multi-class classification problems. Softmax regression is a generalization of logistic regression to multiple classes. In this implementation, we utilize the softmax function to compute class probabilities from the output of the linear model.

By extending the `LinearModel` class, we maintain consistency and reusability across different types of models.

Logistic Regression Outputs:

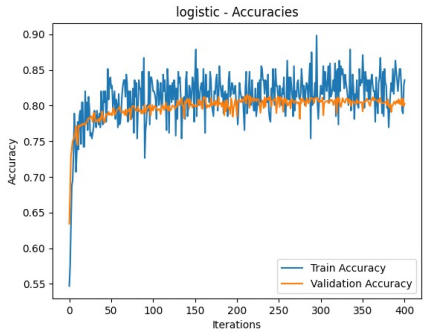
Listing 2: Command for Logistic

```
nishat@WellsFargo:~/ML_assignment_final$ python3 run.py 23754 \
--mode logistic \
--train_data_path data/cifar10_train.npz \
--test_data_path data/test_images.npz \
--num_iters 4000 \
--lr 0.005 \
--batch_size 256 \
--l2_lambda 1e-3 \
--grad_norm_clip 4
```

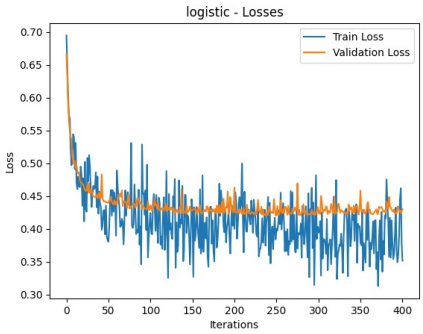


(a) Logistic Accuracy (b) Logistic Loss

Figure 2: Logistic Regression: batch size: **64**, num iter: **4000**



(a) Logistic Accuracy



(b) Logistic Loss

Figure 3: Logistic Regression: batch size: **256**, num iter: **4000**

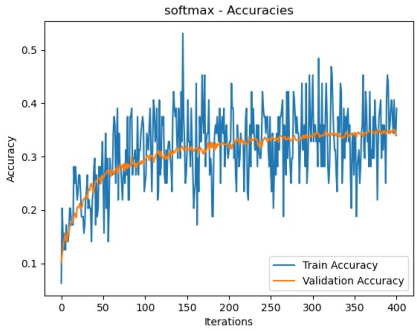
Iter 4000
- Train Loss: 0.3516 - Train Acc: 0.8359 - Val Loss: 0.4302 - Val Acc: 0.8030

Softmax Regression Outputs:

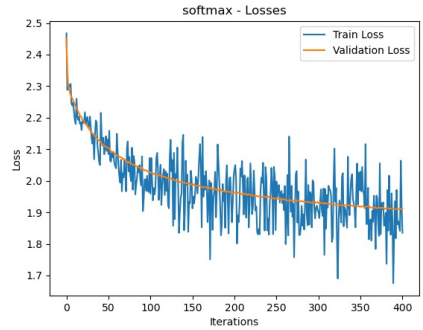
Listing 3: Command for Softmax

```
nishat@WellsFargo:~/ML_assignment_final$ python3 run.py 23754 \
--mode softmax \
--train_data_path data/cifar10_train.npz \
--test_data_path data/test_images.npz \
--num_iters 4000 \
--lr 0.005 \
--batch_size 32
--l2_lambda 1e-3 \
--grad_norm_clip 4
```

Outputs For Softmax Regression:



(a) Softmax Accuracy



(b) Softmax Loss

Figure 4: Output for Softmax Regression

Iter 4000
- Train Loss: 1.6479 - Train Acc: 0.4375 - Val Loss: 1.8270 - Val Acc: 0.3684

2 Problem 2: Contrastive Representation

2.1 Defination

Contrastive representation learning is a technique used in machine learning to learn representations of data that emphasize the differences between different data points. The goal is to learn a representation where similar data points are mapped close together in the representation space, while dissimilar data points are mapped far apart.

One common approach to contrastive representation learning is to use a siamese network architecture. In a siamese network, two identical neural networks share the same parameters, and each network takes as input a different data point. The output of the networks is then compared in a contrastive loss function, which encourages similar data points to have similar representations and dissimilar data points to have dissimilar representations.

Mathematically, the contrastive loss function can be defined as follows:

$$\mathcal{L}(x_i, x_j, y) = \begin{cases} -\log\left(\frac{e^{f(x_i)\cdot f(x_j)/\tau}}{\sum_{k=1}^{N} e^{f(x_i)\cdot f(x_k)/\tau}}\right) & \text{if } y = 1 \\ -\log\left(1 - \frac{e^{f(x_i)\cdot f(x_j)/\tau}}{\sum_{k=1}^{N} e^{f(x_i)\cdot f(x_k)/\tau}}\right) & \text{if } y = 0 \end{cases} \tag{2}$$

where x_i and x_j are input data points, $f(\cdot)$ is the neural network function, y is the label indicating whether x_i and x_j are similar ($y = 1$) or dissimilar ($y = 0$), and τ is a temperature parameter that controls the sharpness of the contrastive loss.

By optimizing the parameters of the siamese network to minimize the contrastive loss, we can learn representations that capture the underlying structure of the data and are useful for downstream tasks such as classification or clustering.

This class encapsulates the logic necessary for binary classification tasks.

$$P(y = k|\mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{e^{\mathbf{w}_k^T \mathbf{x} + b_k}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^T \mathbf{x} + b_j}} \tag{3}$$

where:

- $P(y = k|\mathbf{x}; \mathbf{W}, \mathbf{b})$ is the probability that $\mathbf{x}$ belongs to class k,
- $\mathbf{W}$ is the weight matrix,
- $\mathbf{b}$ is the bias vector,
- $\mathbf{w}_k$ is the k-th row of $\mathbf{W}$, and
- b_k is the k-th element of $\mathbf{b}$.

We can visualize this just the way we do in Metric learning specifically Learning to mearsure, we have two classes say for similar data point we have set S and for dissimilar datapoints we have set D. We fix a anchor data point (blue dot) and we push the negative data point (red dot) farther away from anchor and positive data point (green dot) closer to our anchor. Figure 5 shows how our data points will transform after operation. We learn to measure the distance between points to make our Hypothesis space mor discriminatory. In this space now we can visualize our points better and use this highly discriminatory space to solve complex task.

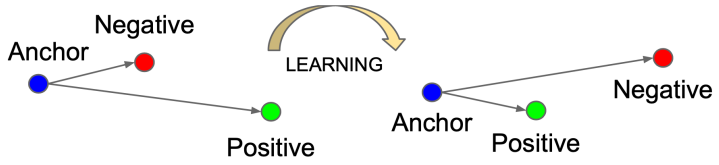


Figure 5: Positive and Negative Data points

To visualize this we say there is a parameter called margin (radius) as you can see in Figure 6 below, we have 2 imposters in local neighbourhood of this anchor data point x, so all the data points from Class 1 lying in this margin are pulled with Epsilon Pull and imposter data points are pushed by Epsilon Push.

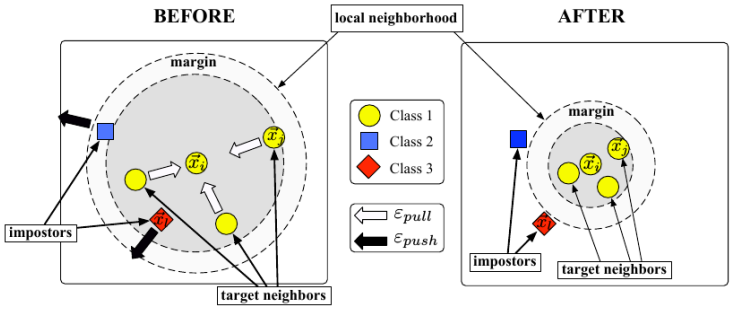


Figure 6: Margin

One of the techniques to do this is to use Mahalanobis distance (distance metric) to measure the distance between a point and a distribution. It is a Euclidean distance with a covariance matrix ,it must have the properties of nonnegativity, the identity of indiscernibles, symmetry, and the triangle inequality. M needs to be symmetric and positive semidefinite. All of the eigenvalues or determinants of M must be positive or zero to be positive semidefinite. The concept of deep metric learning has been shown in Figure 7. For this assignment I have use NLLLoss (Negative log likelihood) for loss calculation. When our feature vectors are in One hot encoding, cross entropy becomes negative log likelihood. NLL Loss serves as an effective optimization objective for training classification models, helping them learn to assign high probabilities to the correct classes and low probabilities to incorrect classes.

Definition: For a classification problem with C classes, the NLL Loss for a single example x_i with true label y_i and predicted probability distribution $\mathbf{p}_i$ can be defined as:

$$\text{NLLLoss}(\mathbf{p}_i, y_i) = -\log(\mathbf{p}_i[y_i])$$

Here, $\mathbf{p}_i[y_i]$ denotes the probability assigned by the model to the true class label y_i.

Explanation: The NLL Loss penalizes the model more severely when it assigns low probability to the true class label. This encourages the model to produce higher probabilities for the correct class and lower probabilities for incorrect classes.

I have used it in combination with a softmax activation function in the output layer of the neural network. The softmax function converts the raw output scores into a

probability distribution over classes, and the NLL Loss is then applied to compare this distribution with the true labels.

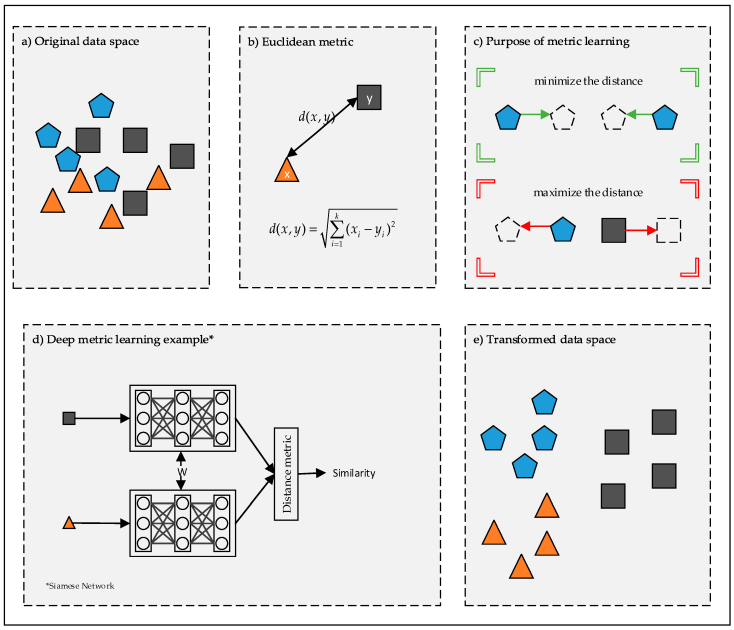


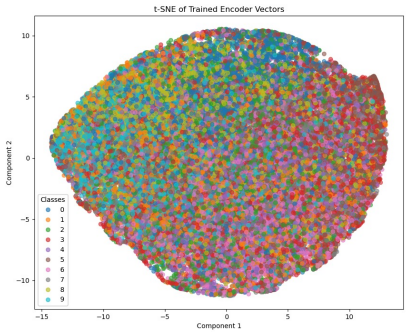
Figure 7: Deep Metric Learning

2.2 Comparative Outputs

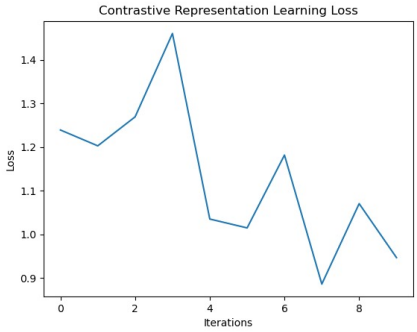
Below is comparision of different outputs of contrastive representation with changing parameters.

Listing 4: Command for Contrastive Representation

```
python3 run.py 23754 \
--mode cont_rep \
--num_iters 1200 \
--lr 1e-3 \
--batch_size 1024 \
--l2_lambda 0.0005 \
--grad_norm_clip 10
```

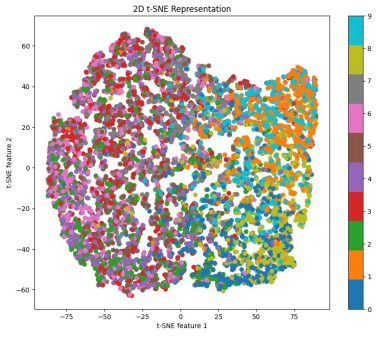


(a)

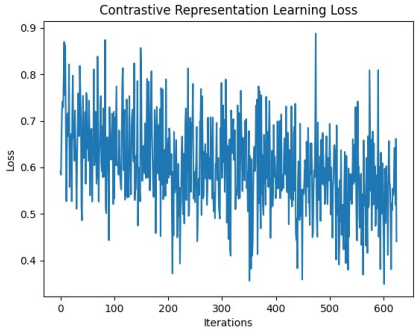


(b)

Figure 8: **Iteration: 10, batch size: 32**

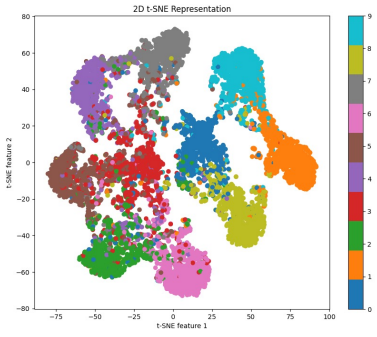


(a)

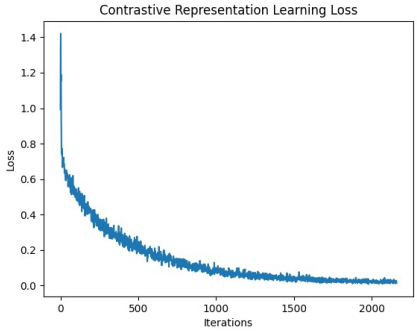


(b)

Figure 9: **Iteration: 800, batch size: 64**

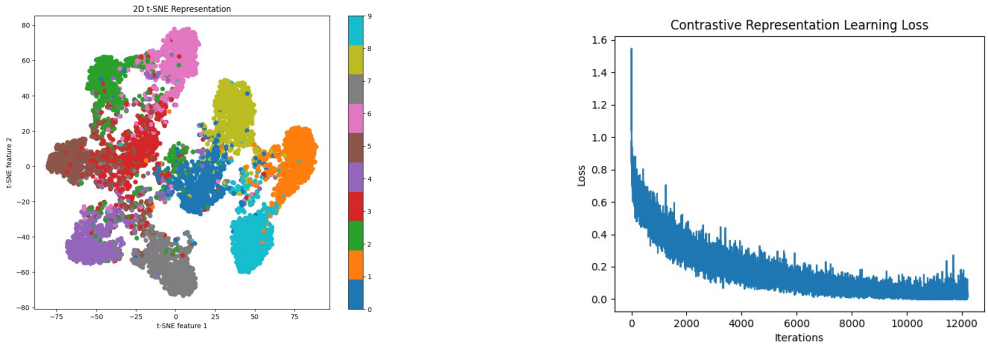


(a)



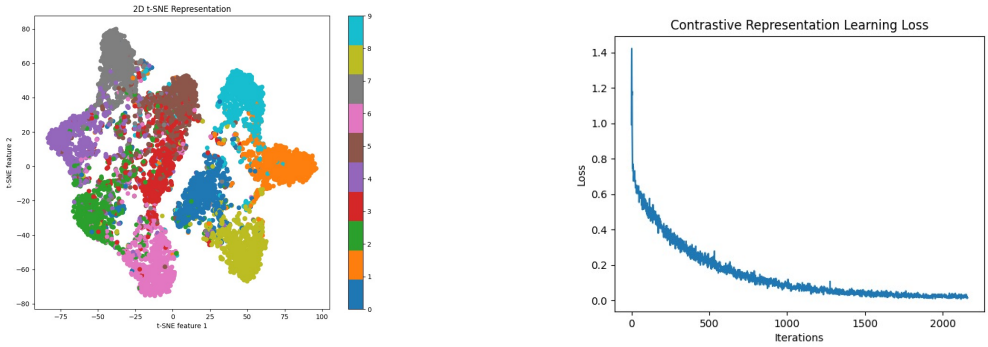
(b)

Figure 10: **Iteration: 1200, batch size: 1024**



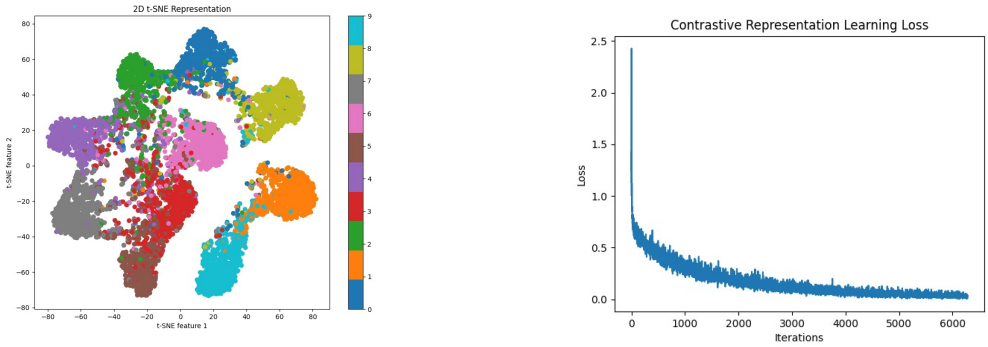
(a) (b)

Figure 11: **Iteration: 1500, batch size: 128**



(a) (b)

Figure 12: **Iteration: 1500, batch size: 256**



(a) (b)

Figure 13: **Iteration: 1800, batch size: 256**

2.3 Fine tune Linear

When args.mode is set to 'fine_tune_linear', the script initializes an encoder model and a linear classifier. The encoder's pre-trained weights are loaded, and the data is converted into PyTorch tensors for training. After training the linear classifier, predictions are generated for the test data using the encoder representations.

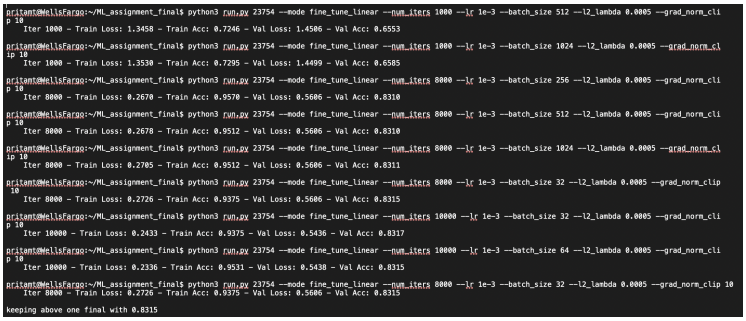


Figure 14: **num_iters 8000, batch_size 32**

Iter 8000
- Train Loss: 0.2726 - Train Acc: 0.9375
- Val Loss: 0.5606 - Val Acc: 0.8315

2.4 Fine tune nn

In the 'fine_tune_nn' mode, the script aims to adapt a neural network classifier to the specific features extracted by the pre-trained encoder. By utilizing the encoder's representations as input features for the classifier, the model can use the encoded information to make more informed predictions. It helps to learn task-specific patterns from the encoded data, potentially enhancing its classification performance.

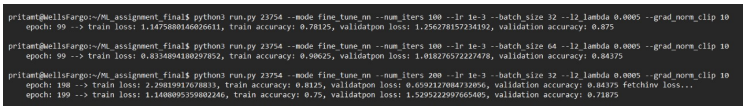


Figure 15: **num_iters 100, batch_size 32**

Epoch: 99
- Train Loss: 1.1475880146026611 - Train Acc: 0.78125
- Val Loss: 1.256278157234192, Val Acc: 0.875

3 Refrences

- **Deep Metric Learning: A Survey** https://www.mdpi.com/2073-8994/11/9/1066
- **Pattern Analysis and Applications** https://www.researchgate.net/A-comparison-of-ORS-learning-with-distance-metric-learning-The-ORS-tries-to-pull$_fig1_336711414$
- **PyTorch Metric Learning: An opinionated review** https://www.pento.ai/blog/pytorch-metric-learning
- **Deep Dive into Softmax Regression** https://towardsdatascience.com/deep-dive-into-softmax-regression-62deea103cb8