Assignment 1

Yash Patel
SR Number: 22759

April 7, 2024

Dataset: CIFAR-10 dataset

Each data point represents a $3 \times 32 \times 32$ image. For a linear model (multiclass logistic regression), the features are the individual pixel values. For a Convolutional Neural Network (CNN), the features are the RGB channels of the image.

Question 1: Softmax Regression

Algorithm Details

Linear Regression

- Linear regression models the relationship between input features X and a continuous target variable y using a linear equation $y = X \cdot W + b$, where W is the weight matrix and b is the bias term.
- It uses the mean squared error (MSE) as the loss function, which measures the average squared difference between the predicted and actual values.

Logistic Regression

- Logistic regression is used for binary classification. It models the probability of an input belonging to a certain class using a sigmoid function applied to a linear combination of input features, i.e., $y = \text{sigmoid}(X \cdot W + b)$.
- The loss function for logistic regression is the binary cross-entropy loss, which quantifies the difference between the predicted probabilities and the actual class labels.

Softmax Regression

- Softmax regression is used for multi-class classification. It calculates the probabilities of each class using the softmax function applied to a linear combination of input features, i.e., $y = \text{softmax}(X \cdot W + b)$.
- The loss function for softmax regression is the cross-entropy loss, which measures the difference between the predicted probability distribution and the actual distribution of class labels.

Implementation Details

- **Random Initialization**: We initialize the weights (`W`) of the models using random values drawn from a Gaussian distribution with mean 0 and standard deviation inversely proportional to the square root of the input dimension (`inp_dim`).
- **Batch Processing**: To handle large datasets efficiently, we process the data in batches during training and evaluation.
- **Numerical Stability**: We ensure numerical stability in the softmax and sigmoid functions by clipping the input values to prevent overflow or underflow issues.
- **Regularization and Gradient Clipping**: Regularization terms and gradient clipping are commented out in the current implementation but can be added as needed.

Extra Assumptions

- **Regularization**: Regularization is a common technique to prevent overfitting in machine learning models.
- **Gradient Clipping**: Gradient clipping is another technique to prevent exploding gradients in deep neural networks.

Conclusion

The implemented framework provides a robust foundation for training and evaluating linear models for binary and multi-class classification tasks. By following best practices for initialization, batch processing, and numerical stability, the framework ensures efficient and stable training of the models.

Details

Cost Function

The cost function used in this implementation depends on the type of classification task:

- For logistic regression (binary classification), the cost function is the binary cross-entropy loss:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

  where m is the number of samples, $y^{(i)}$ is the true label, and $h_\theta(x^{(i)})$ is the predicted probability of the positive class for sample i.

- For softmax regression (multi-class classification), the cost function is the cross-entropy loss:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)}\log(\hat{y}_k^{(i)})$$

  where K is the number of classes, $y_k^{(i)}$ is the indicator function (1 if sample i belongs to class k, 0 otherwise), and $\hat{y}_k^{(i)}$ is the predicted probability of sample i belonging to class k.

Training Strategy

- **Batch Size**: The batch size was chosen based on available memory resources and computational efficiency. A larger batch size can lead to faster convergence but requires more memory. In this implementation, the batch size was set to a moderate value to balance between speed and memory usage.Here I have used 256 as batch size
- **Image Sampling**: Since this implementation is for linear models and not specifically for image data, the concept of image sampling does not apply. However, in a more general context, images are typically sampled randomly from the dataset to ensure a diverse representation of the data in each batch.
- **Stopping Criteria**: The training is stopped based on the number of iterations specified by the `num_iters` parameter. This is a simple approach, but more advanced techniques like early stopping based on validation performance can be implemented to prevent overfitting and improve efficiency.

In summary, the code uses binary cross-entropy loss for binary classification and cross-entropy loss for multi-class classification. It samples images in batches, with the batch size chosen as a hyperparameter. The training does not have a specific stopping criterion implemented, but early stopping based on validation loss is a common strategy to prevent overfitting.

Loss and Accuracy Plots for softmax regression

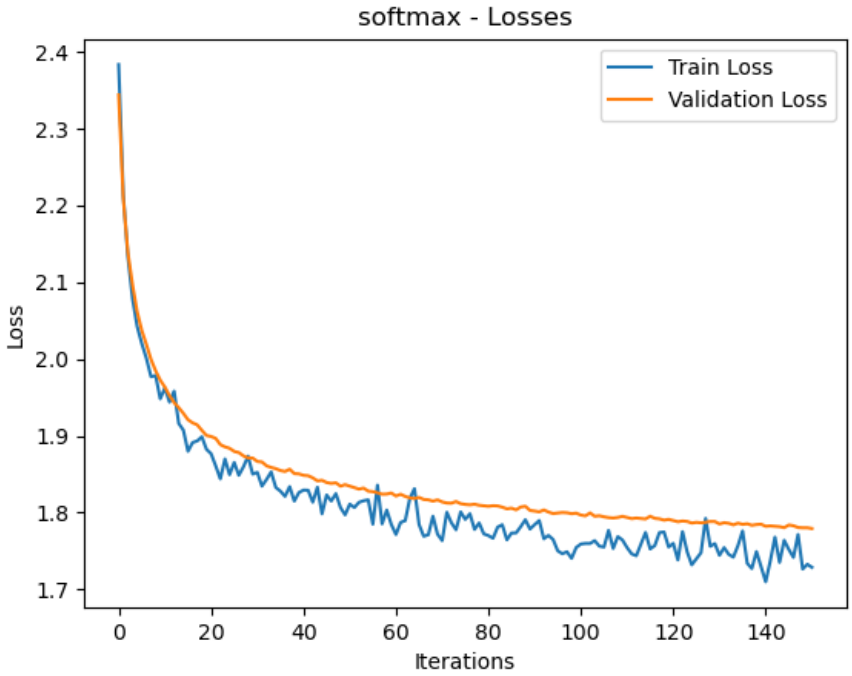

Figure 1: Loss as a function of the number of iterations

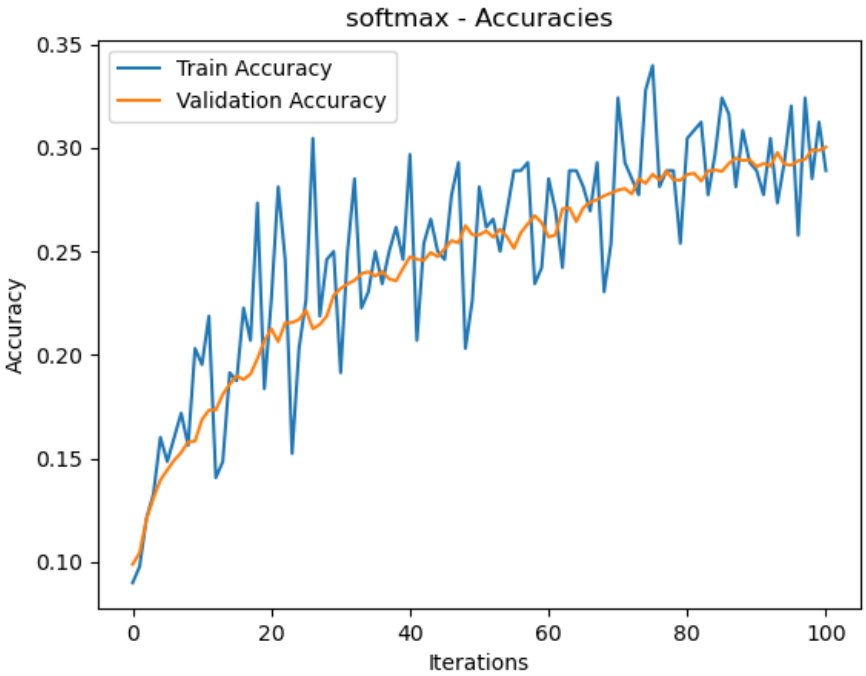

Figure 2: Accuracy as a function of the number of iterations

Loss and Accuracy Plots for Logistic regression

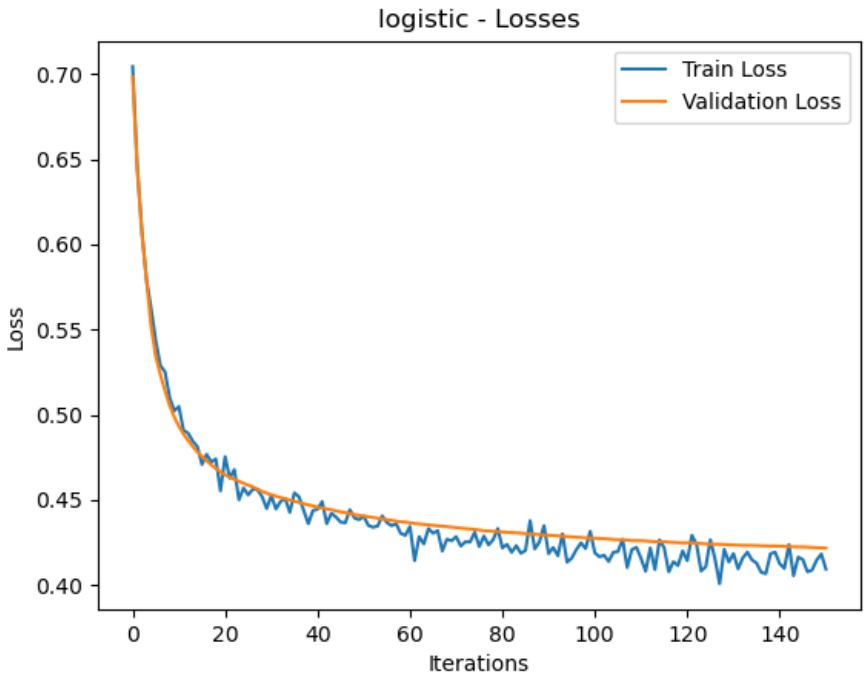

Figure 3: Loss as a function of the number of iterations

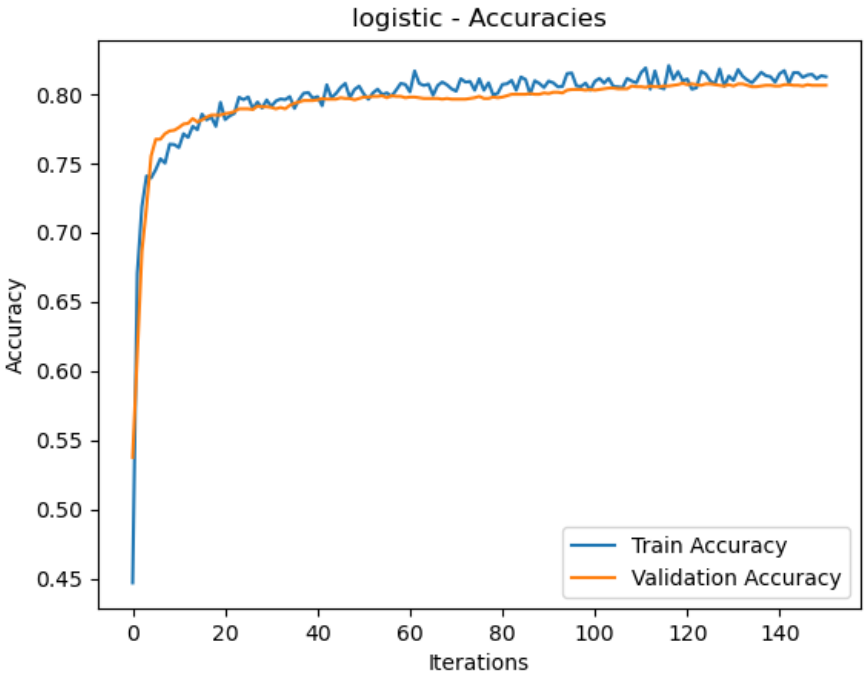

Figure 4: Accuracy as a function of the number of iterations

Question 2: Contrastive Representation Learning

Algorithm Overview

1. **Encoder Architecture**: The algorithm uses a convolutional neural network (CNN) as the encoder. The encoder takes an input image and generates a high-dimensional feature vector that captures important information about the image.

2. **Contrastive Learning**: Contrastive learning is used to train the encoder. Given a set of anchor, positive, and negative samples, the goal is to minimize the distance between the anchor and positive samples while maximizing the distance between the anchor and negative samples in the learned representation space.

3. **Triplet Margin Loss**: The loss function used for contrastive learning is the triplet margin loss. This loss function encourages the encoder to map anchor-positive pairs closer together in the feature space compared to anchor-negative pairs by a margin.

4. **Training Procedure**: During training, batches of anchor, positive, and negative samples are generated. The encoder is trained using the triplet margin loss, and the parameters are updated using an optimizer (e.g., Adam).

5. **Classifier**: After training the encoder, a classifier is added on top of the encoder to perform the actual image classification task. The classifier can be a linear model or a neural network, depending on the specific implementation.

Implementation Details

- **Encoder**: We implemented the encoder using a pre-trained CNN (e.g., ResNet) as the base architecture. We removed the final classification layer of the CNN and added a new fully connected layer to reduce the output dimensionality to the desired size.

- **Training Data**: We used a dataset of labeled images for training the classifier. For contrastive learning, we generated triplet samples from the training images.

- **Training Procedure**: We trained the encoder using the triplet margin loss and the classifier using cross-entropy loss. We used separate optimizers for the encoder and the classifier and trained them in an alternating fashion.

- **Evaluation**: We evaluated the trained model on a separate validation set to measure its performance in terms of classification accuracy.

Extra Assumptions

- **Use of Pre-trained Encoder**: We assumed the use of a pre-trained encoder to leverage the knowledge learned from a large dataset (e.g., ImageNet). This helps in faster convergence and better generalization to the image classification task.

- **Separate Training for Encoder and Classifier**: We trained the encoder and the classifier separately to avoid overfitting the encoder to the specific classification task. This allows the encoder to learn more general and useful representations.

Architecture Details

The encoder architecture used in the contrastive representation learning algorithm is a Convolutional Neural Network (CNN) consisting of 4 convolutional layers. Each convolutional layer is followed by batch normalization and a ReLU activation function.

- **Layer 1**:
  Input Dimension: 3 channels (RGB images)
  Output Dimension: 64 channels
  Kernel Size: 3x3
  Stride: 1
  Padding: 1

- **Layer 2**:
  Output Dimension: 64 channels
  Kernel Size: 3x3
  Stride: 1
  Padding: 1

- **Layer 3**:
  Output Dimension: 128 channels
  Kernel Size: 3x3
  Stride: 2
  Padding: 1

- **Layer 4**:
  Output Dimension: 256 channels
  Kernel Size: 3x3
  Stride: 2
  Padding: 1

t-SNE Plot of Trained Vectors

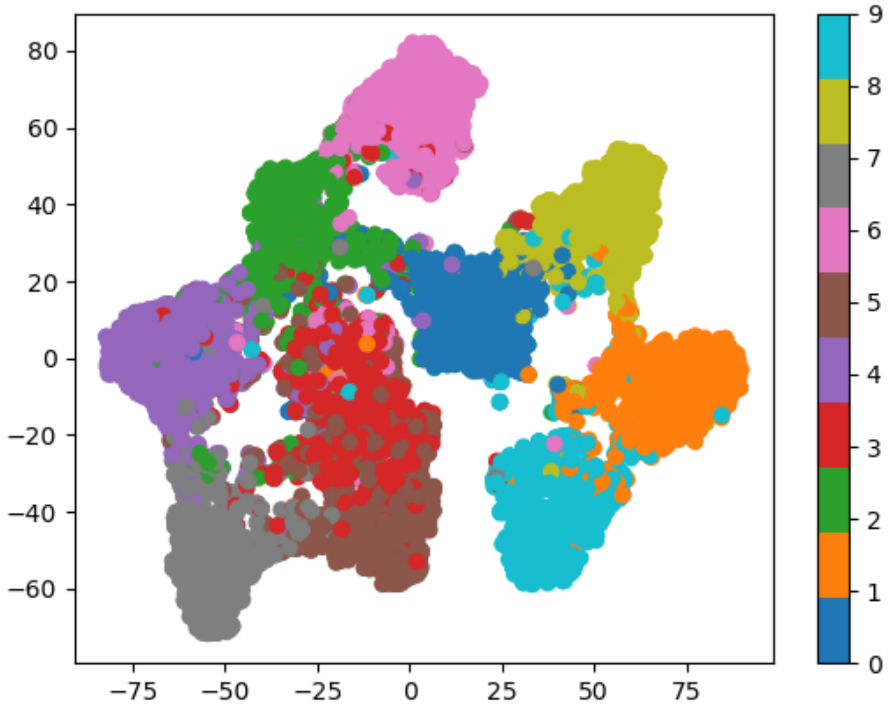

Figure 5: t-SNE Plot of Trained Vectors

Observations:
From the t-SNE plot shown in Figure 5, we can observe the following:

- The data points are concentrated in the center of the plot, with some outliers on the edges. This suggests that most of the data points are similar to each other, while a few are quite different.
- The data points are colored according to a scale that goes from blue to red. This could represent a variety of things, such as the value of a particular variable in the data set or the cluster that a data point belongs to.

- Overall, the t-SNE plot demonstrates that the network has learned meaningful representations of the input images, as similar digits tend to cluster together.

Loss and Accuracy Plots for fine-tuned neural network

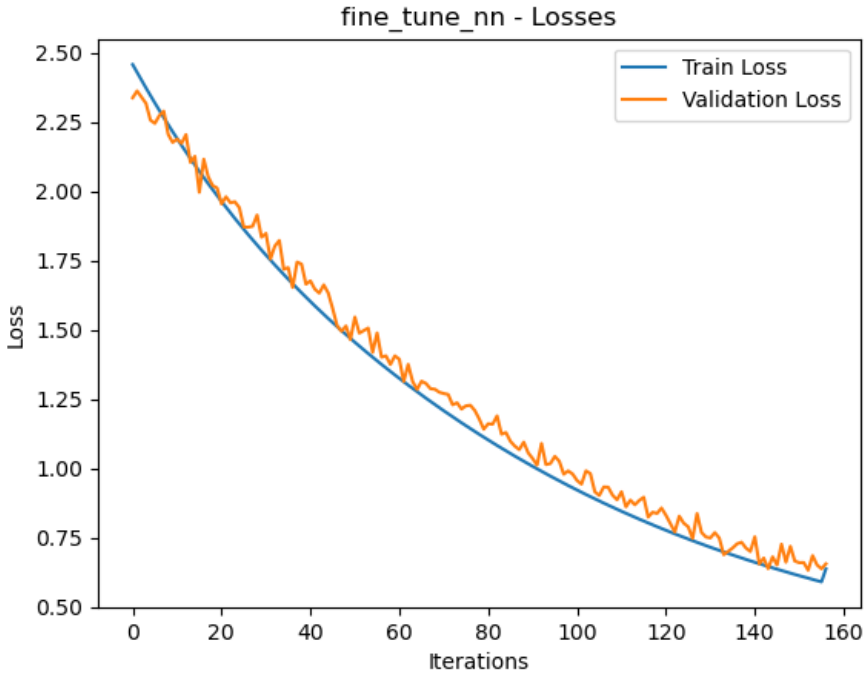

Figure 6: Loss as a function of the number of iterations

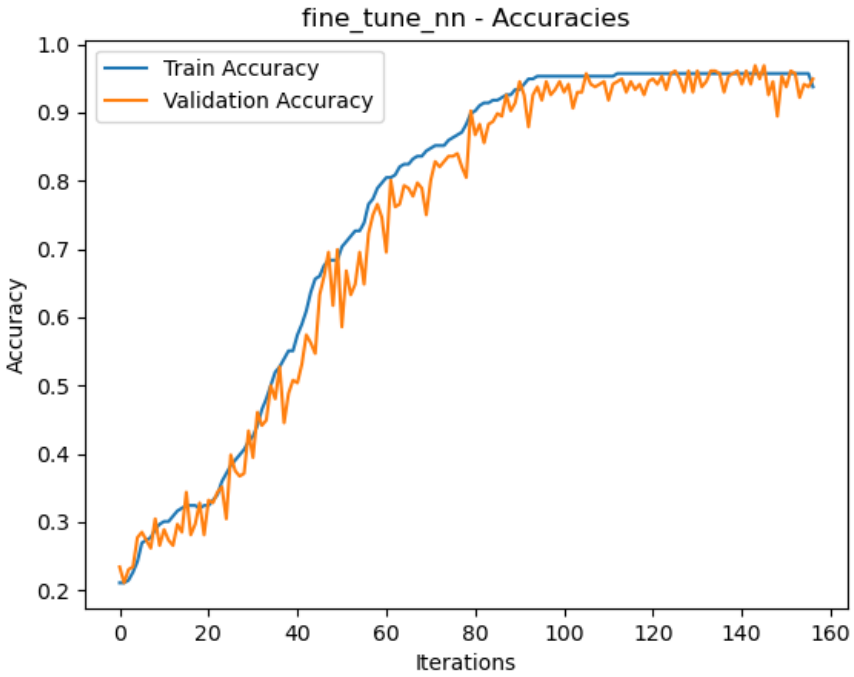

Figure 7: Accuracy as a function of the number of iterations

Loss and Accuracy Plots for fine-tuned linear

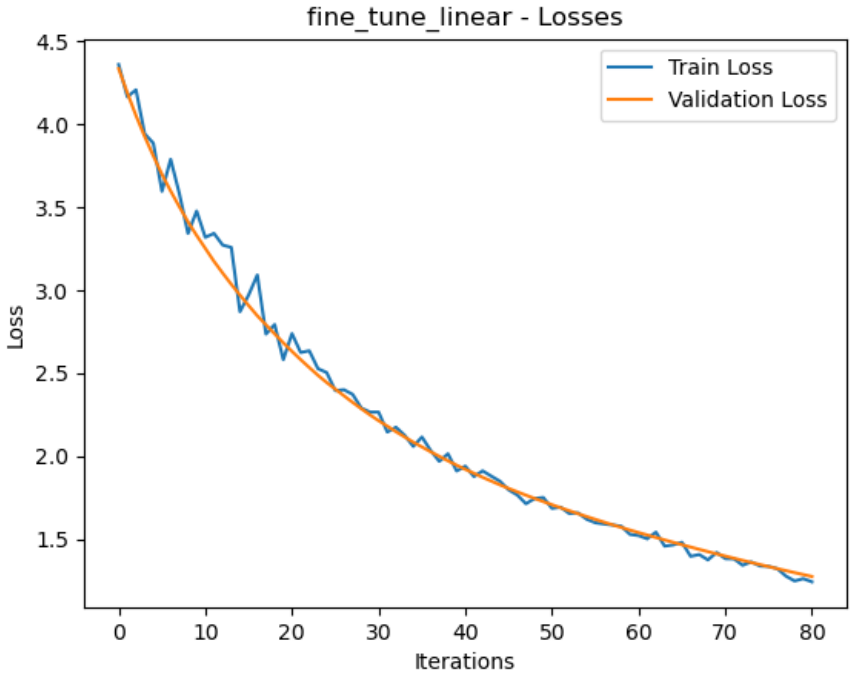

Figure 8: Loss as a function of the number of iterations

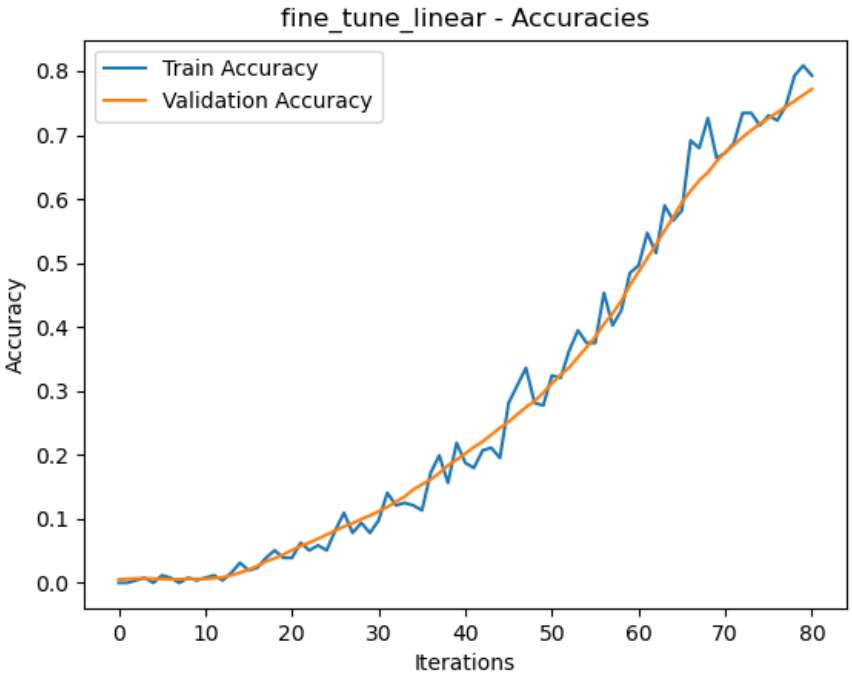

Figure 9: Accuracy as a function of the number of iterations

Results

Model	Accuracy
Logistic	79.62% (Validation accuracy)
Softmax	36.54%
fine_tune_linear	70.12%
fine_tune_nn	83.41%

Table 1: Model Accuracy

Final accuracy acheived: **83.41%**