Neural Networks and Deep Learning - Coursework: CIFAR-10 Classification

Task 1: Reading Dataset and Creating Data Loaders

Environment Setup:

Firstly, the code is configured to be device agnostic thereby if GPU is available then its CUDA cores are used for faster computation else CPU is used. To reproduce the results in any environment the random seed is set to 42.

Data Preparation:

The transformations for the train and test dataset are defined separately initially.

- Train Transformations contains Random horizontal flipping with 50% probability and random cropping with padding for augmentation, and normalization using mean and standard deviation values of (0.5, 0.5, 0.5)
- Then, image, which typically has pixel values in the range [0, 255] is converted, into a PyTorch tensor with values normalized to [0, 1]. This transformation changes the image data from HxWxC format (Height x Width x Channels) to CxHxW (Channels x Height x Width), which is the input format expected by PyTorch convolutional layers.
- Test Transformations only converts the images to tensors and applies the same normalization as the training dataset to ensure consistency.

Next, The CIFAR-10 dataset is downloaded and loaded into training and testing datasets respectively. Finally, they are passed to be set up in DataLoaders which splits the train and test datasets into separate batches with a batch size of 64. The training loader shuffles data to prevent sequence learning bias.

Task 2: Creating the Basic & Advanced Model Architectures

Basic Model - Architecture:

The basic model was built as per the given architecture requirements. This consists of a class IntermediateBlock which defines a series of blocks within which a series of convolutional layers and other layers are present. The final output from these blocks is then passed to a OutputBlock which acts as a classifier layer to compute the class scores.

- **Intermediate Block:** This block is responsible for feature extraction from the input data. It consists of a series of block within which multiple convolutional layers, each followed by batch normalization (to improve training stability) and a ReLU activation function are present. In this block, first the mean of each channel are computed and passed to a fully connected layer to obtain the weights. Meanwhile the input image passed parallelly to each convolutional layer in a block is multiplied with the weights obtained previously and this output is passed to other sequential layers in the same block. The output of this block is sent to the next sequential block which has the same number of layers.
- **Output Block:** This takes the final output from the final blocks as input. It computes the mean of the feature maps along the spatial dimensions (channels) and passes them through multiple fully connected layers, present in the final layer that outputs the logits corresponding to each class.
- **Base Model Class:** This class combines multiple intermediate blocks followed by an output block to perform image classification.

Advanced Model - Architecture:

The Advanced Model builds upon the Basic Model by modifying the IntermediateBlock to include adaptive pooling, more convolutional layer arrangements and hyper parameter tuning of the basic model parameters.

The new features added in the advanced model are as mentioned below:

- Adaptive average pooling is used to compute the mean of each channel.
- Leaky ReLu activation function is used instead of ReLu which avoids vanishing gradient and sparsity problem
- A max pooling layer is added additionally which down samples the output passing to the next layer

The basic model was hyper-parameter tuned to achieve improvement in the model accuracy. The hyper parameters of each model are displayed below.

Features	Basic Model	Advanced Model
Number of Blocks	4	5
Convolutional Layers in Each Block	4	5
Hidden Units in Each Layer in Intermediate Blocks	10	512
Number of fully connected layers in output block	2	4

Task 3: Choosing Best Suited Loss Function and Optimizer

Cross Entropy Loss Function: It is designed for multi-class classification problems. We make use of cross-entropy loss in our model training since it considers the probabilities through which our model predicts for each class. It penalizes the model more heavily for assigning low probabilities to the correct class, and less heavily for assigning high probabilities to incorrect classes.

AdamW Optimiser: I used the AdamW optimizer because it deals better with weight decay and, hence, generalization of the model. It is also one of the regularization technique that helps to prevent overfitting. In our multi-classification, overfitting can occur when the model learns the training data too well and fails to generalize to unseen data. Therefore, Weight decay has helped to prevent overfitting by forcing the model to have smaller weight.

Task 4: Training & Testing of the Model

The code defines two functions, **training_step** and **testing_step**, essential for training and evaluating a neural network model. The training_step function is responsible for adjusting the model's parameters based on the training data to minimize the loss function, while the testing_step function evaluates the trained model's performance in terms of loss and accuracy on unseen data. The accuracy function is also defined to calculate the accuracy scores.

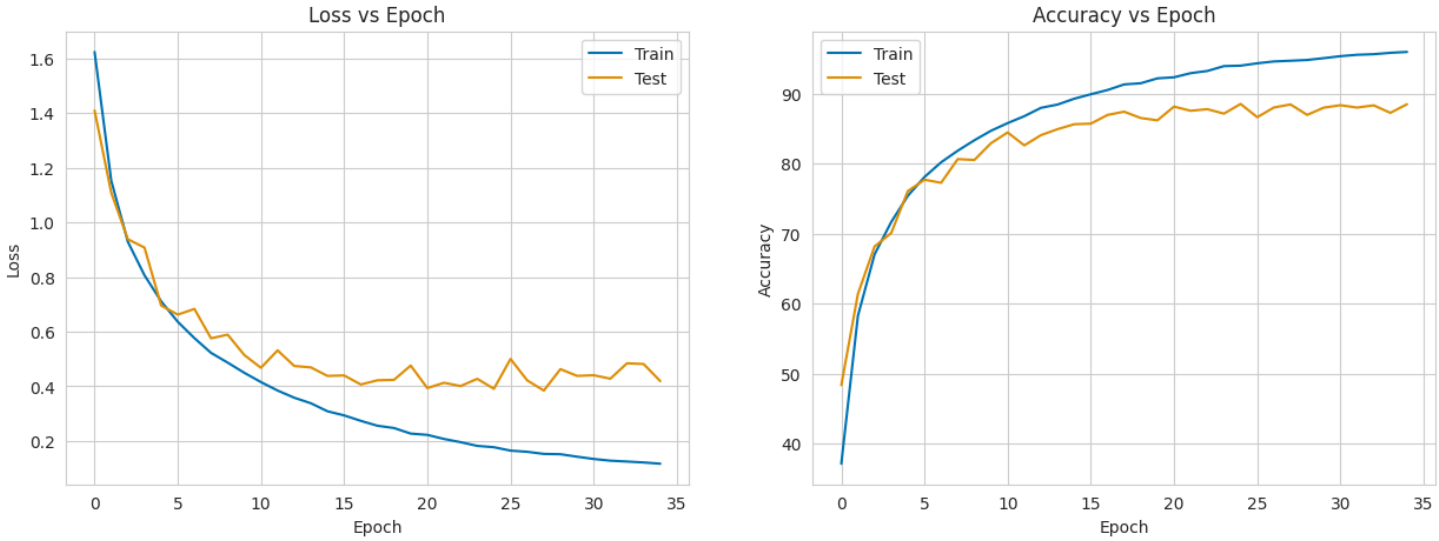
Train Process:

1. Moves the model to the appropriate device (CPU or GPU) and sets it to train mode which performs the forward pass of the inputs to the model and obtains the prediction logits.
2. Computes the cross-entropy loss between the predictions and the true labels and reset (zeroes) the optimizer's gradients to prevent accumulation from previous iterations.
3. Performs a backward pass to compute gradients w.r.t the loss and updates the model's weights using the optimizer's step function.
4. Calculates and prints the average loss and accuracy over all batches.

Test Process:

1. Moves the model to the appropriate device (CPU or GPU) and Sets the model to evaluation mode to disable gradients and dropout layers for consistent predictions.
2. Initializes variables for accumulating total loss and accuracy and Disables gradient calculations using `torch.inference_mode()` for efficiency during evaluation.
3. Iterates over batches of data, moving each batch to the device, and performs a forward pass to generate predictions.
4. Computes and accumulates the loss and accuracy for each batch, then calculates and prints the average loss and accuracy over all batches.

Finally, after iterating over 35 epochs, the code returns the list of training and test loss and accuracy values for each epoch which will be used to plot the results.



Advanced Model Loss & Accuracy for Train vs Test

From the above loss and accuracy graphs, it is evident that the model is doing well on the CIFAR-10 dataset. Both training loss and training accuracy with each epoch are decreasing and increasing, respectivel. This means the model is learning and hence improving while being trained. Here, the test loss goes down, and the test accuracy increases as each epoch goes through, again another good sign that the model is generalizing well to new data.

Below are the accuracy metrics of the Basic and Advanced models for train and test data.

Model-Dataset	Loss	Accuracy
Advanced-Train	0.11675	95.98
Advanced-Test	0.41915	88.51
Basic-Train	0.94936	65.76
Basic-Test	1.101544	64.10

The model was trained for 35 epochs. After the final epoch, the basic model achieved a maximum training accuracy of 65.76% and a maximum test accuracy of 64.10%. The advanced model achieved a maximum training accuracy of 95.98% and a maximum test accuracy of 88.51%.

Conclusion:

We developed two CIFAR-10 image classification models: a basic and an advanced version. The advanced model built upon the basic structure, with added layers and hyperparameter tuning, enhancing its accuracy and generalization capabilities for new data.

[Note: Due to page limitations, detailed explanations, basic model's and references are attached in the notebook]