ECS7026P - Neural Networks and Deep Learning Coursework

By – Satyam Sharma (220760793)

Task 1 - Read datasets and create dataloaders

To enhance the effectiveness of the model in recognizing new images, we employ data augmentation techniques on the training set. These techniques involve applying various transformations, such as -

1. RandomResizedCrop: Randomly crop and resize images with padding.
2. RandomRotation: Randomly rotate images up to 15 degrees.
3. RandomHorizontalFlip: With a 50% chance, the images are horizontally flipped.
4. ToTensor: Convert images to PyTorch tensors.
5. Normalize: Normalize images by subtracting the mean (0.5) and dividing by the standard deviation (0.5) for each color channel.

The validation set, on the other hand, undergoes only the conversion to tensor and normalization transformations to maintain the original images. Both the training and validation datasets are then loaded into DataLoader instances with a batch size of 64, enabling batching, shuffling, and parallel processing during both training and testing phases.

Task 2 - Create the Model

Model Architecture - The ResNet-based model is composed of a backbone and a classifier.

Backbone:

The backbone is a sequence of 7 blocks, each of which consists of convolutional layers, batch normalization, and a PReLU activation function. The purpose of the backbone is to extract relevant features from the input image, by applying convolutional filters and nonlinear transformations. Specifically, each block uses adaptive convolutions, where the weights of the convolutional filters are learned from the input image by applying a fully connected layer to obtain channel-wise weights. These weights are then applied to the convolutional filters to produce the output of the block. Additionally, every other block in the backbone includes a max pooling layer, which reduces the spatial dimensions of the input and increases the number of channels.

Classifier:

The output of the 7th block of the backbone is then passed to the classifier, which consists of a global average pooling layer, a flatten layer, and a fully connected layer with 10 output neurons, corresponding to the 10 possible classes in the classification task. The purpose of the classifier is to take the feature representations learned by the backbone and produce a final classification decision. The global average pooling layer reduces the spatial dimensions of the feature map to 1x1, while preserving the channel information. The flatten layer then converts the feature map into a 1D vector, which is fed into the fully connected layer to produce the final class scores.

Task 3 - Create the loss and optimizer

1. Loss Function:

- A custom loss function called "LabelSmoothingCrossEntropyLoss" is defined to handle label smoothing in the training process.
- The loss function calculates log probabilities of the input tensor using log_softmax and then calculates the cross-entropy loss with element-wise multiplication of weights and log probabilities, summing along the class dimension, and then taking mean value.
- The weight values for the target class and non-target classes are calculated based on the smoothing value and the number of classes.

2. Optimizer:
   - RMSprop optimizer is used to optimize the model parameters.
   - The learning rate is set to 0.001, alpha to 0.99, eps to 1e-08, weight_decay to 0, and momentum to 0.9.
3. Scheduler:
   - A learning rate scheduler defined with cosine annealing is also added.
   - The scheduler reduces the learning rate by gradually decreasing it to eta_min (1e-6 in this case) over T_max (100 in this case) epochs.

Task 4 - Write the training script to train the model

Training Function:

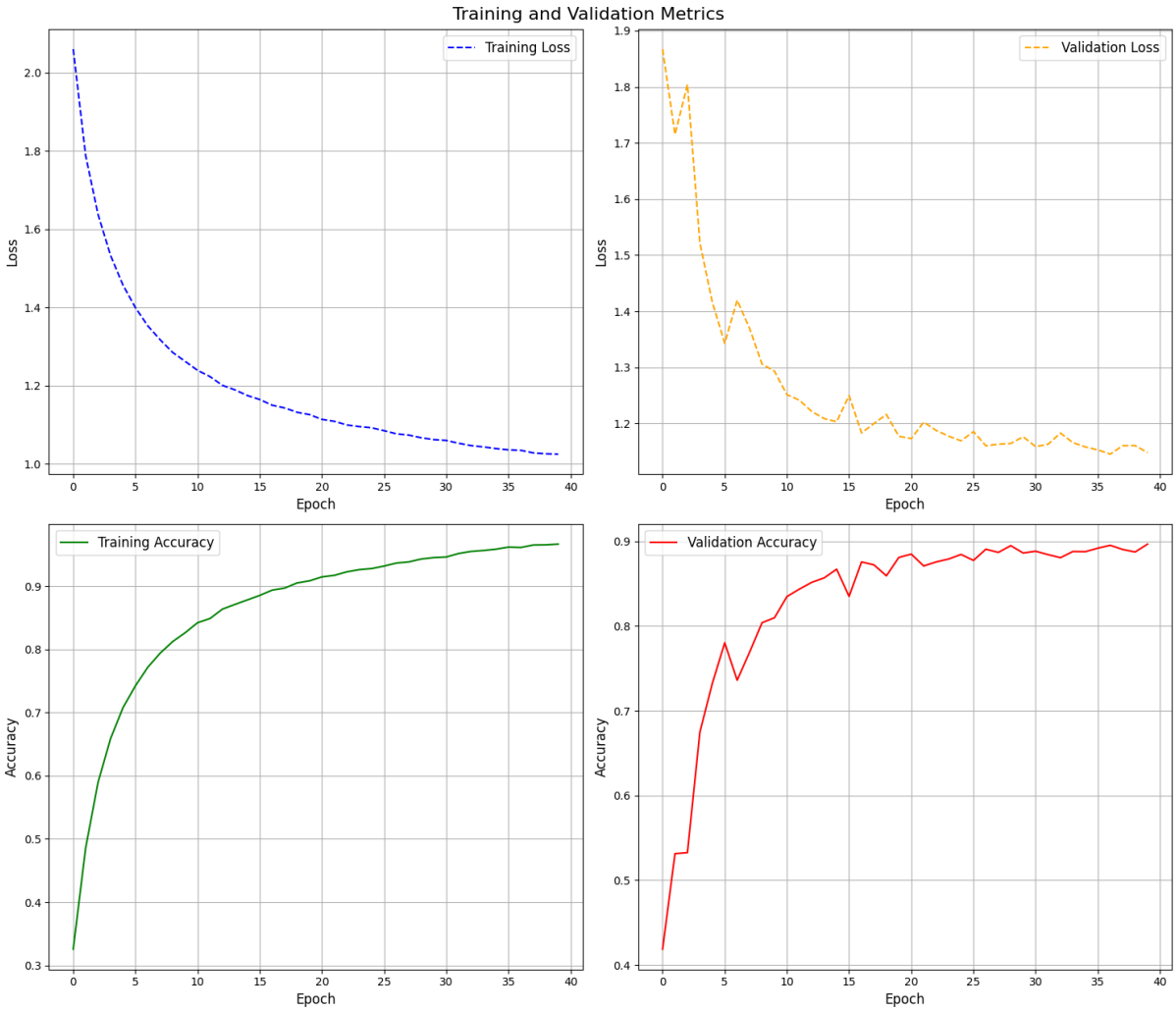
The train_epoch function trains a given neural network model for one epoch using the above dataloader, criterion, optimizer, and device. During training, the function accumulates gradients for a specified number of mini-batches (num_accumulation_steps) before updating the model parameters, which helps to reduce memory usage and speed up training. The function calculates the loss and accuracy for the epoch and returns these values. Additionally, the function sets the model to training mode and moves the data to the specified device.

Validation Function:

The validation_epoch function evaluates a given neural network model for one epoch using a specified dataloader, criterion, and device. It sets the model to evaluation mode and loops through the test data to calculate the loss and accuracy of the model's predictions. For each batch of validation data, the function moves the batch images and labels to the specified device, performs a forward pass through the model using torch.no_grad(), calculates the loss using the specified criterion, and accumulates the loss. It also gets the predicted class labels and counts the number of correct predictions. At the end of the loop, it calculates the average loss and accuracy for the epoch and returns these values.

Training the model:

Here we train and validate the model for 40 epochs, stores the training and validation loss and accuracy history for each epoch, and prints the results at the end of each epoch.
We begin by defining the number of epochs = 40 to train for and initializing empty lists to store the training and validation loss and accuracy history.
Then we use loops to move through each epoch, train the model for one epoch using the train_epoch function, validate the model using the validation_epoch function, and update the learning rate scheduler using the scheduler.step() method.
For each epoch, we store the calculated training and validation losses and accuracies in the corresponding lists. At the end of each epoch, we print the epoch number, the training loss and accuracy, and the validation loss and accuracy.



Task 5 - Final model accuracy on the CIFAR10 Validation Set

The model achieved impressive results with a training accuracy of 96.66% and a validation accuracy of 89.66% for the final epoch. The achieved high accuracy levels indicate that the model is successfully learning and recognizing the underlying patterns present in the data and is able to apply this knowledge to new, previously unseen data. The training loss and validation loss values of 1.0249 and 1.1479 respectively provide evidence of the model's strong performance. Additionally, the small difference between the training and validation metrics suggests that the model is not overfitting, which means it can generalize well to new data. Overall, the model has achieved a good balance between fitting the data and being able to generalize.

These high scores the result of various techniques and steps used to enhance the model's performance. Data augmentation techniques, such as RandomResizedCrop, RandomRotation, and RandomHorizontalFlip, were applied to the training set, while the validation set underwent only normalization and tensor conversion. The ResNet-based CNN model was composed of a backbone and a classifier, with the backbone extracting relevant features from the input image, and the classifier producing the final classification decision. Additionally, a custom loss function and the RMSprop optimizer with a learning rate scheduler were used to optimize the model's parameters. Overall, these steps helped to achieve a well-balanced model that is able to recognize new images with a high degree of accuracy.