Neural Networks and DL: Coursework

Class	ECS7026P
Student	Shahin Mammadov
Student ID	240315623

Implemented Architecture Description:

The implemented neural network architecture, encapsulated within the `CIFAR10Net` and `IntermediateBlock` classes, is directly derived from the specifications outlined in Section 2 of the provided document. The overall structure follows the requirement of stacking a sequence of intermediate blocks (`num_blocks` instances of `IntermediateBlock`) followed by an output mechanism. The core `IntermediateBlock` specifically implements the logic detailed in Section 2.1: it features `num_parallel` (corresponding to L) independent convolutional pathways (`self.convs`), each operating directly on the block's input tensor `x`, fulfilling a key requirement. The block's output is constructed as a weighted sum of these pathways' results, precisely mirroring the formula $x' = \sum a_i C_i(x)$. To achieve this, the attention weights (`a`) are calculated dynamically from the input `x` by first applying global average pooling (to derive the channel average vector m described in the requirements) and then passing this through a fully connected layer (`self.fc`), directly implementing the specified weight generation process. Finally, the classification head within `CIFAR10Net` (global average pooling followed by a linear layer) corresponds to the Output block described in Section 2.2, processing the final feature map to produce the classification logits (o).

1. `IntermediateBlock`:
   - This block implements the core parallel processing idea described in Section 2.1.
   - It takes an input tensor `x` with `in_channels`.
   - It has `num_parallel` (equivalent to L in the description) independent convolutional pathways.
   - **Weight Calculation:** It computes attention weights (`a`) for these pathways. This is done by applying adaptive average pooling to the *input* tensor `x` (reducing spatial dimensions to 1×1, effectively getting the average of each input channel), flattening the result, and passing it through a linear layer (`self.fc`) whose output size is `num_parallel`. A softmax function is applied to normalize these weights.
   - **Parallel Convolutions:** Each of the `num_parallel` pathways consists of a single `nn.Conv2d` layer (using the specified `kernel_size` and `padding`) followed by an activation function (defaulting to ReLU). Crucially, each of these convolutional layers operates on the *original input* `x` of the block. All parallel convolutions output tensors with `out_channels`.
   - **Output:** The final output of the block is a weighted sum of the outputs from the parallel convolutional pathways, where the weights are the computed attention scores `a`. The output tensor has `out_channels`.
2. `CIFAR10Net`:
   - This class defines the overall network structure.
   - It stacks `num_blocks` instances of the `IntermediateBlock` sequentially.
   - The number of input channels for the first block is `in_channels` (e.g., 3 for CIFAR-10). The number of output channels for *all* intermediate blocks is fixed at `out_channels`. The input channels for subsequent blocks match the output channels of the preceding block.
   - **Classification Head:** After the final `IntermediateBlock`, the network applies adaptive average pooling to the feature map (reducing spatial dimensions to 1×1), flattens the result, and passes it through a single final linear layer (`self.fc`) to produce the logits for the `num_classes`.

Training and Testing

Each model is using the Cross-Entropy loss (`nn.CrossEntropyLoss()`), and is evaluated on **20 epochs**. Throughout the training and evaluation process, the code maintains several lists to record performance metrics:

- `epoch_train_loss`: Records the average training loss per epoch.
- `epoch_train_accuracy`: Captures the training accuracy computed after processing all training batches in each epoch.
- `epoch_test_accuracy`: Stores the test accuracy measured at the end of each epoch.
- `all_batch_losses`: Keeps a detailed record of the loss for each batch within every epoch. This can be used later for deeper analysis of convergence and potential fluctuations during training.

We will first start with experimental setup, after which we will make mild changes to the architecture and then do hyperparameter search. At the end, we will see how each step improved the output.

Model #1: Experimental Setup

Hyperparameters chosen for this first experiment are:

- **Output Channels:** 32 per intermediate block.
- **Number of Intermediate Blocks:** 4.
- **Number of Parallel Convolution Layers per Block:** 4.

- **Kernel Size:** 3.
- **Padding:** 1.
- **Optimizer:** Adam
  - lr: 0.001
- **Convolutional Layer:** Conv2d → activation

Model #2: Modifications to Intermediate Block and Output

Each block processes the input through multiple parallel branches. The branches have been enhanced by deepening their structure with a two-stage sequence:

Intermediate Block

- **Stage 1:** Convolution, followed by batch normalization and an activation function.
- **Stage 2:** A second convolution, again followed by batch normalization and the activation function.

Output

- After global average pooling and the flattening of the feature map, dropout is applied immediately before the final linear layer. As we now have deeper structure, `dropout` is added to prevent potential overfitting.

Hyperparameters chosen for this first experiment are:

- **Output Channels:** 32 per intermediate block.
- **Number of Intermediate Blocks:** 4.
- **Number of Parallel Convolution Layers per Block:** 4.
- **Kernel Size:** 3.
- **Padding:** 1.
- **Optimizer:** Adam
  - lr: 0.001
- **Convolutional Layer:** Conv2d → BatchNorm2d → activation → Conv2d → BatchNorm2d → activation

Model #3: Hyperparameter Optimization

The following hyperparameters, spanning architecture, optimization, and data settings, will be tuned together in this search:

Architecture Parameters:

- `num_blocks`: Number of IntermediateBlocks to stack
- `num_parallel`: Number of parallel paths/branches in each block
- `out_channels`: Number of output channels for convolutional layers
- `kernel_size`: Size of the convolution kernels
- `padding`: Padding size for convolutions

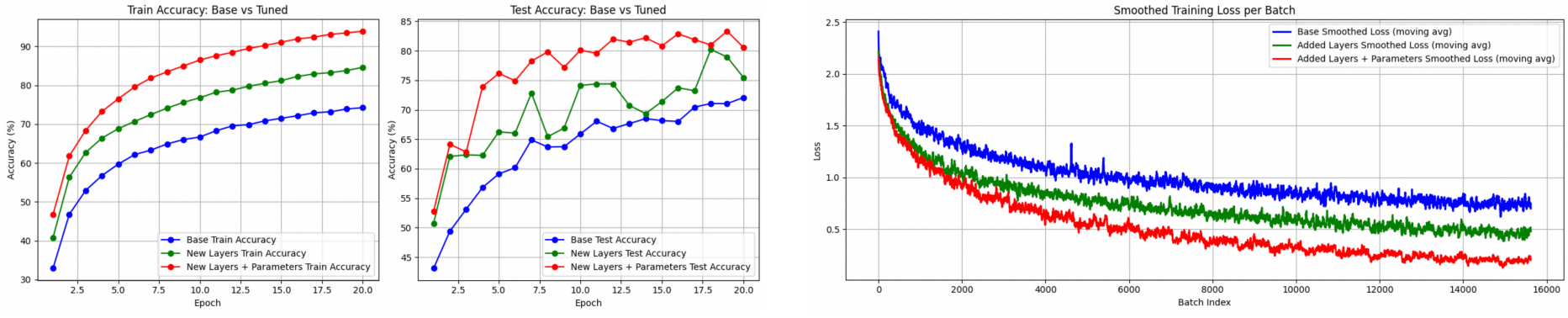
Optimization Parameters:

- `lr`: Learning Rate
- `weight_decay`: L2 Regularization strength
- `optimizer_name`: Choice of optimizer

We explored combinations of these parameters over the 40 trials, aiming to find the configuration that yields the best performance (e.g., highest validation accuracy) on the modified base architecture. These are the results:

- **Output Channels:** 64 per intermediate block.
- **Number of Intermediate Blocks:** 5.
- **Number of Parallel Convolution Layers per Block:** 3.
- **Kernel Size:** 5.
- **Padding:** 1.
- **Optimizer:** Adam
  - lr: 0.00012970850431931997
  - weight_decay: 0.00036101432184583877
- **Convolutional Layer (not changed):** Conv2d → BatchNorm2d → activation → Conv2d → BatchNorm2d → activation

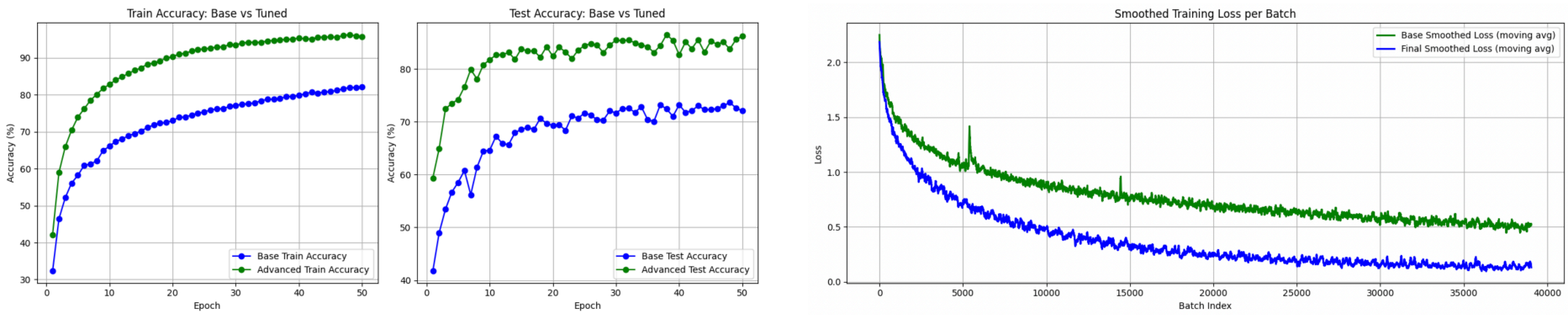
Improvements



We can see a clear improvements in every next step we took. The blue line represents the experimental setup, with no additional layers inside the intermediate block and hyperparameter tuning. When **new layers** were introduced, they provided enhanced representational capacity—enabling deeper feature extraction—and thus pushed both training and test accuracies higher than the baseline. Finally, adding **further parameter refinements** (or additional tuning) built on the benefits of the deeper architecture and led to even more robust learning, resulting in higher accuracy on both the training and testing curves and signaling improved generalization.

Final Evaluation

After tuning our model, we evaluated both the base model and advanced model, using 50 epochs, where we will evaluate the accuracy of both models.



We can see that the final model performs much better than the base model, where we have incorporated additional layers inside intermediate block, to extract finer details, also hyperparameter search, which found the optimal parameters, such as number of blocks, number of parallel convolutional layers, kernel size, output and padding. All of that combined, we can see a jump from 72% accuracy on test data, to 86% accuracy.

Model-Dataset	Accuracy
Base-Train	82.2%
Base-Test	72%
Advanced-Train	95.7%
Advanced-Test	86.2%