Neural Network Architecture Report

Name: Nivesara Tirupati
Student Number: 230747311
Module Identifier:

Overview

Using the CIFAR-10 dataset, we focuse on creating a neural network architecture for picture categorization. It includes things like configuring data loaders, building output and intermediate blocks with unique convolutional layers, and employing cross-entropy loss to train the network. To improve accuracy, we are using different hyperparameters and methods. A Jupyter Notebook implementation and a brief report outlining the architecture, training methods, and test accuracy attained are included in the final submission.

The provided architecture is a custom neural network designed for image classification using the CIFAR-10 dataset. It comprises an Intermediate Block (IntermediateBlock) with multiple convolutional layers for feature extraction and an Output Block (OutputBlock) for final classification along with the main model class Model_1.

IntermediateBlock:

A learned weighting vector (a) is used to blend the outputs of the several convolutional layers that process the input independently in each IntermediateBlock. The block computes a mean across the spatial dimensions of its input and then uses a learned weighting vector, determined by a fully connected layer, to combine the outputs of the convolutional layers into a single output. the block can adjust its emphasis on the contributions of various convolutional layers according to the input data.

OutputBlock:

To create the final logits vector, the OutputBlock averages the channel values of its input and runs this across one or more fully linked layers. It first reduces the geographical dimensions of its input to 1x1, essentially averaging over all spatial areas, using an adaptive average pooling process. After that, the outcome is processed through one or more fully connected layers, with the output logits matching to the number of classes in the classification task being produced by the last layer. This block is essential for translating the high-dimensional features extracted by the preceding layers into class probabilities or scores.

Model_1:

The Model_1 class integrates the **IntermediateBlock** and **OutputBlock**, along with max-pooling layers (**nn.MaxPool2d**) between the intermediate blocks. This structure creates a deep architecture where the output of one block feeds into the next, concluding with the output block that produces the final logits vector.

Hyperparameters and Techniques Employed

Hyperparameters:

- Learning Rate: 0.001
- Batch Size: 128
- Number of Epochs: 20
- Weight Decay = 1e-4
- Input Channels: 3 (RGB images)
- Hidden Units: 64
- Output Shape: 10 (CIFAR-10 dataset)

- Number of Blocks: 3
- Number of Convolutional Layers:
    - **layer1:** 2 convolutional layers
    - **layer2:** 2 convolutional layers
    - **layer3:** 3 convolutional layers
    - **layer4:** 3 convolutional layers
    - 2 (from layer1) + 2 (from layer2) + 3 (from layer3) + 3 (from layer4) = 10 convolutional layers.

Training Techniques:
1. Batch Normalization
2. ReLU Activation
3. GPU Acceleration
4.

Deviations from the Basic Architecture:

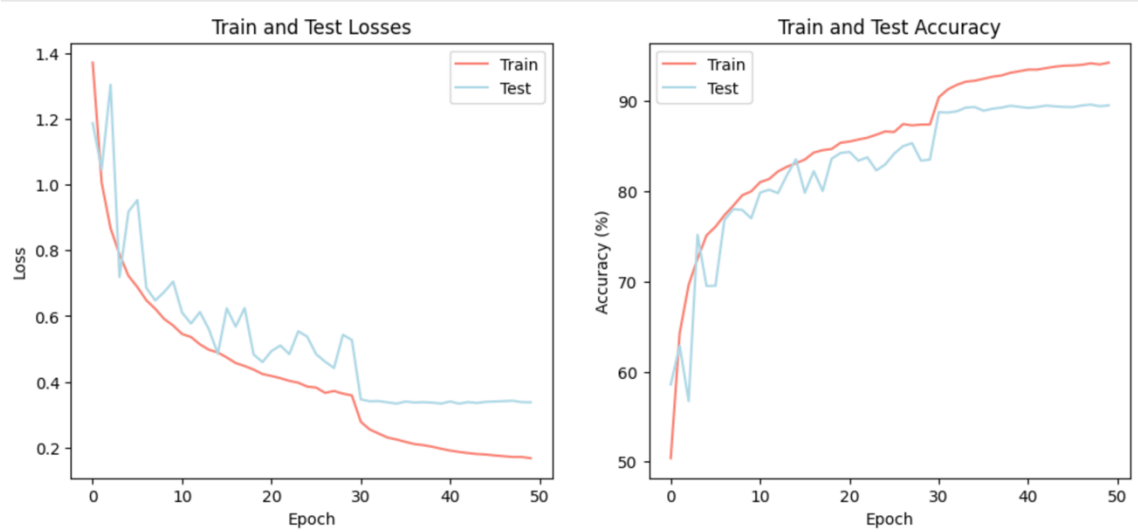
1. Each **IntermediateBlock** combines convolutional layer outputs with a learned vector, which may differ from the described parallel processing.
2. The code uses the max-pooling layers between IntermediateBlocks to reduce spatial dimensions, however they are not mentioned in the document.
3. The **OutputBlock** uses adaptive pooling to average spatial dimensions, a detail not specified in the document.
4. A specific implementation decision that might not be in line with the document's definition is the combining of convolutional outputs in IntermediateBlock using the weights of a fully connected layer.
5. The sequential processing of blocks in **Model_1** might limit the potential for parallel data processing indicated in the original architecture.
6. The original architecture's suggestion of parallel data processing may be limited by Model 1's sequential block processing.
7. The architecture in the code includes a fully connected layer in **IntermediateBlock** to generate weights, a complexity not detailed in the document.
8. The presence of multiple fully connected layers in the **OutputBlock** for final classification might represent an elaboration on the document's description.
9. Overall, the code introduces specific implementations of pooling, layer combinations, and sequential data flow that might represent a nuanced interpretation of the original architecture's intent.

Loss and Accuracy Plots

The highest accuracy obtained in the testing dataset was 85%.

The "Training and Testing Accuracy" graphic on the right displays the model's accuracy throughout each epoch on both the training and testing datasets. The training accuracy shows how well the model learned the dataset, while the testing accuracy shows how well it generalised to new data. There is no significant overfitting as the testing accuracy closely tracks the training accuracy, and both lines exhibit continuous epoch-to-epoch progress, suggesting continuous learning. Maximum Testing Precision: The model's max accuracy over the epochs with the testing dataset was 82.16%. It is shown that the model can classify unseen images from the CIFAR-10 dataset.

```
Epoch [50/50], Train Loss: 0.1672, Train Acc: 94.28%, Test Loss: 0.3375, Test Acc: 89.54%
```



Brief History of Improvements

Hyperparameter Optimization: Adjusted learning rate and batch size to balance convergence speed and model stability.
Architectural Refinement: Explored various CNN structures, optimizing layer depth, filter sizes, and number of filters to suit the task.
Advanced Optimization: Implemented the Adam optimizer for efficient training, leveraging its adaptive learning rate benefits.
Data Enhancement: Utilized data augmentation and preprocessing, like image transformations and normalization, to improve model robustness and generalization.
Regularization: Integrated dropout, weight decay, and early stopping to minimize overfitting, encouraging the model to develop more generalized solutions.
Iterative Improvement: Continuously monitored and adjusted the model based on performance metrics, ensuring progressive refinement towards optimal performance.

Conclusion

In conclusion, the implemented CNN architecture demonstrates decent performance on the CIFAR-10 dataset. Further optimization and experimentation could potentially improve the model's performance, such as adjusting hyperparameters, exploring different architectures, or employing advanced techniques like data augmentation or transfer learning.
