Pranav Rao

Student ID: 240837228

Course ID: ECS7026P

Neural Networks and Deep Learning

Dynamic CNN for CIFAR-10 Image Classification

This project implements a dynamic convolutional neural network architecture for image classification on the CIFAR-10 dataset. The architecture works by dynamically combining multiple convolutional layers within intermediate blocks to improve classification accuracy. This report will outline the neural network architecture, discuss hyperparameters and training techniques, and include visualizations that outline the loss for each training batch and the training and test accuracy for each epoch. Optimization techniques will also be outlined. Finally, the highest obtained testing accuracy will be displayed.

Architecture Outline

The neural network architecture is composed of four intermediate blocks that feed into a final output block. Each intermediate block receives an input image tensor and processes it through four parallel convolutional layers. However, instead of each input image being fed through sequentially, each convolutional layer receives the same input image, and the outputs from these convolutional layers are dynamically combined through a weighted summation. To calculate these dynamic weights, the architecture computes the channel wise average of the input features, resulting in a compact feature vector. This vector is then passed through a fully connected layer, producing coefficients corresponding to each convolutional layer. These coefficients are then normalized using a softmax function for proper weighting.

The four intermediate blocks incrementally increase the number of output channels, specifically following the sequence of 64, 128, 256, and 512 channels. Each convolutional layer within these blocks incorporates Batch Normalization (BatchNorm) and Rectified Linear Unit (ReLU) activation functions and are initialized using Kaiming initialization as it tends to be effective to layers with ReLU activation. The inclusion of BatchNorm and ReLU after each convolutional layer is a deliberate deviation from the basic architecture described in Section 2, intended to enhance training stability and improve model performance. In addition, another deviation from the basic architecture is the addition of residual connections, which adds the original input to the weighted sum of convolutional outputs. This design choice, inspired by ResNet architectures, helps mitigate vanishing gradients and improve training stability.

The output block takes the processed feature map from the final intermediate block and computes its channel wise average (global average pooling). This is then processed through two fully connected layers, each equipped with BatchNorm, ReLU activations, and dropout regularization (with dropout rates of 0.4 and 0.3, respectively). Finally, the output block produces a logits vector with ten units, corresponding to the CIFAR-10 classification task.

Key deviations introduced to achieve higher accuracy and better generalization include the incorporation of BatchNorm and ReLU after each convolution, the use of dropout regularization in the output block, the expansion of the network depth to four intermediate blocks, and a max pooling operation in between intermediate blocks 2 and 3.

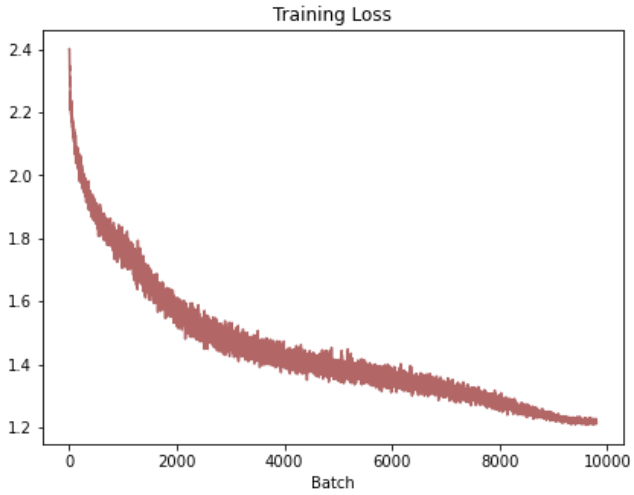
Hyperparameters and Training Techniques

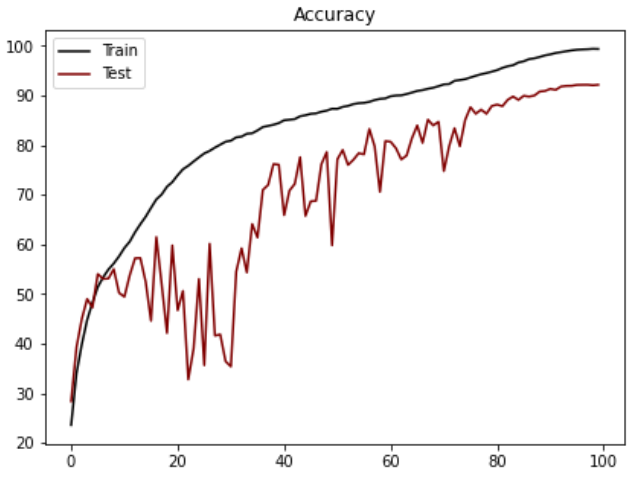
Parameter	Value
Optimizer	SGD (Nesterov)
Learning Rate	0.15
Momentum	0.95
Weight Decay	1e-3
Batch Size	512
Label Smoothing	0.3
Epochs	100
Gradient Clipping	0.5

To further enhance the network's performance, a few advanced training techniques were also employed. Comprehensive data augmentation strategies were used, including random cropping (32×32 pixels with padding of 4), horizontal flipping, random rotations within 15 degrees, and color jittering. These augmentations expanded the training dataset size and improved the network's ability to generalize beyond the provided images. Additionally, regularization methods including dropout in the fully connected layers of the output block, weight decay, and label smoothing helped reduce the risk of overfitting by encouraging the network to learn more generalized feature representations. The learning rate was set at 0.15 because it worked better with the implemented OneCycle scheduler, which was chosen because of the faster initial learning, allowing it to skip the local minima and ramp up training more efficiently. Optimization was further accelerated and stabilized through mixed precision training with PyTorch's autocast functionality, allowing faster computations and reduced memory consumption. The combination of dynamic learning rate scheduling, gradient clipping, extensive data augmentation, and multiple regularization techniques played a critical role in achieving the final high classification accuracy, enabling the model to robustly generalize to unseen data.

Plots

Following the architecture and training sections, the model was run and data on training loss and accuracy were collected. This information was then summarized into a plot and is displayed below:





The highest test accuracy achieved by this model was **92.15%.** This is backed up by the plots, shown above. The training loss plot shows the loss decreasing slowly over the course of training, which is what should be expected from a properly functioning model. The accuracy plot shows the training accuracy increasing steadily, which means the model is learning from the training data well. While the test accuracy was more volatile in the beginning due to the high learning rate with OneCycle scheduler, it does have a general upward trend and has no major dips or fluctuations as training neared the end. In addition, this was somewhat rectified after the implementation of residual connections, which stabilized training slightly. With both a high training accuracy and a high test accuracy, the model has no major risk of overfitting and may generalize well to unseen data.

Steps taken to Improve Results

Initially, the neural network closely followed the basic architecture and achieved a baseline performance of 72%. To enhance accuracy, several key improvements were made. First, implementing dynamic learning rate scheduling via the OneCycle policy accelerated convergence. Next, extensive experimentation with data augmentation techniques, including random rotations, cropping, and color jitter, led to accuracy improvements by improving the model's robustness. Introducing Batch Normalization and ReLU activations after every convolutional layer greatly stabilized training, allowing the use of higher learning rates. Furthermore, integrating mixed precision training using PyTorch's autocast accelerated computations and allowed more effective utilization of GPU resources. Finally, regularization methods such as dropout in the output block and label smoothing reduced overfitting and improved generalization. These incremental adjustments, carefully tested and integrated, culminated in a substantial improvement from the initial implementation to the final accuracy achieved (**92.15%**).