Custom Deep Residual Network for CIFAR-10

Aashna Kunkolienker, Akshita Upadhyay
NetIDs: ank8919, apu2005
NYU Tandon School of Engineering
Project Codebase: `https://github.com/aashnakunk/Dl_Mini_project.git`

Abstract

This project explores the task of image classification on the CIFAR-10 dataset using a custom ResNet architecture. The goal is to train a model to accurately classify images into one of ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The custom ResNet architecture includes data augmentation techniques such as random horizontal flip, random crop, color jitter, and random rotation to improve the model's generalization ability. The model is trained using stochastic gradient descent with momentum and weight decay, and the learning rate is adjusted using a step scheduler. Experimental results demonstrate the effectiveness of the proposed approach in achieving decent classification accuracy on the test set.

ResNet Model Overview

ResNet Architecture

The ResNet architecture consists of multiple blocks of basic blocks, each followed by downsampling layers to reduce the spatial dimensions. The number of blocks and the number of channels in each block can be adjusted based on the desired network depth and complexity

The ResNet (Residual Network) architecture is a deep learning model that addresses the vanishing gradient problem in deep neural networks. It introduces skip connections, or shortcuts, that allow the gradient to flow more directly through the network, enabling training of very deep networks.

The ResNet implemented here consists of several blocks of convolutional layers, each followed by batch normalization and ReLU activation. The blocks also include optional dropout layers for regularization. Additionally, a Squeeze-and-Excitation (SE) block is included, which adaptively recalibrates channel-wise feature responses by explicitly modeling interdependencies between channels.

Custom ResNet Model Architecture

The custom ResNet model is designed to leverage the benefits of residual connections and Squeeze-and-Excitation (SE) blocks to improve feature learning and classification performance. The architecture consists of several key components:

- **Initial Convolutional Layer:** The model starts with a 3x3 convolutional layer with 32 filters applied to the input image. This layer is followed by batch normalization and a ReLU activation function to introduce non-linearity.

- **Residual Blocks:** The ResNet architecture uses residual blocks to address the vanishing gradient problem. Each residual block contains two 3x3 convolutional layers with batch normalization and ReLU activation functions. A shortcut connection is added to skip these layers if the input and output dimensions are different.

- **Squeeze-and-Excitation (SE) Block:** Each residual block includes an SE block, which consists of an adaptive average pooling layer followed by two fully connected layers with ReLU activation functions. This block recalibrates channel-wise feature responses, enhancing important features and suppressing irrelevant ones.

- **Layer Stacks:** The model includes four stacks of residual blocks. The number of output channels in each stack increases progressively (32, 64, 128, and 256), while the spatial dimensions are reduced through strides (1, 2, 2, and 2, respectively). Each stack contains multiple residual blocks, with the first block adjusting the number of channels and spatial dimensions if necessary.

- **Global Average Pooling:** After the final layer stack, global average pooling is applied to reduce the spatial dimensions of the feature maps to a single value per channel. This step helps in reducing the computational complexity and makes the model more robust to spatial translations of the input.

- **Fully Connected Layer:** Finally, a fully connected layer is used to map the extracted features to the output classes. The number of output units in this layer corresponds to the number of classes in the classification task (e.g., 10 for CIFAR-10).

The custom ResNet model architecture combines the strengths of residual connections and SE blocks to achieve state-of-the-art performance in image classification tasks. The use of batch normalization and ReLU activation functions throughout the network helps in stabilizing and accelerating the training process.

.

Training Overview

The training process for the custom ResNet model involves several key components and techniques to optimize model performance and stability.

- **Model Initialization:** The ResNet model is initialized and moved to the specified device (e.g., GPU) for training.

- **Optimizer Selection:** Stochastic Gradient Descent (SGD) with momentum is chosen as the optimizer. SGD with momentum helps accelerate convergence and smooth out the oscillations in the training process.

- **Learning Rate Scheduler:** A cosine annealing learning rate scheduler is employed to adjust the learning rate during training. This scheduler helps the model explore different areas of the loss landscape, potentially escaping local minima.

- **Loss Function:** The CrossEntropyLoss function is used as the loss criterion for training the model. This loss function is suitable for multi-class classification tasks.

- **Gradient Clipping:** Gradient clipping is applied to prevent the gradients from becoming too large, which can lead to unstable training and exploding gradients. This technique helps stabilize the training process.

- **Training Loop:** The model is trained over multiple epochs. In each epoch, the model is set to training mode, and the training data is passed through the model. The optimizer is used to update the model parameters based on the calculated loss.

- **Validation:** After each epoch, the model is evaluated on a separate validation dataset to monitor its performance on unseen data. This helps prevent overfitting and allows for early stopping if the model starts to overfit.

- **Best Model Saving:** The best performing model (based on validation accuracy) is saved to a file ('best_model.pth') for future use.

- **Monitoring and Reporting:** The training loop reports the training loss and accuracy after each epoch, as well as the validation accuracy. This allows for monitoring the model's progress and performance throughout training.

Overall, these techniques and components work together to train the custom ResNet model effectively, optimizing its performance and stability while preventing overfitting.

Methodology

Our approach to designing and training the models involved several key decisions and techniques aimed at improving performance and efficiency. Initially, we experimented with different optimizers and settled on Stochastic Gradient Descent (SGD) instead of Adam due to its better convergence properties for our task.

The baseline model we started with had approximately 0.6 million parameters and achieved an accuracy of 88% on the test dataset. However, we observed that the model was not capturing complex features effectively. To address this, we introduced an additional layer to our ResNet architecture, specifically adding the following layer: `self.layer4 = self.make_layer(256, 3, stride=2)`, which increased the number of parameters to 4 million. This modification significantly improved the model's performance, leading to a notable increase in accuracy.

In addition to architectural changes, we incorporated several training techniques to enhance model stability and convergence. One such technique was gradient clipping, which helps prevent the exploding gradient problem by limiting the magnitude of gradients during backpropagation. Gradient clipping proved to be particularly effective in stabilizing the training process and preventing divergence.

During the training phase, we initially experimented with training for 250 epochs. However, we observed that the train and validation accuracy plateaued after a certain number of epochs, indicating that the model had converged. As a result, we decided to reduce the number of epochs to 200, which provided a good balance between training time and performance.

Overall, our methodology involved a combination of architectural adjustments, optimizer selection, and training techniques aimed at improving the performance and efficiency of our models. These decisions were guided by experimentation and iterative refinement, leading to a final model that achieved our desired level of accuracy and stability.

Results

This custom resnet model uses 4398890 parameters.

The training process was conducted over 200 epochs, and the model achieved a peak validation accuracy of 91.27% at the end of the training. The training accuracy steadily increased from 38.53% in the first epoch to 99.39% in the final epoch. Similarly, the validation accuracy improved from 47.00% to 91.27% over the same period.

The training and validation losses decreased consistently throughout the training process, indicating that the model

was learning effectively. The training loss decreased from 1.6460 in the first epoch to 0.0190 in the final epoch.

The final test accuracy was 90.74

Overall, the results demonstrate that the ResNet model was able to effectively learn from the training data and generalize well to unseen data, achieving a high level of accuracy on the CIFAR-10 dataset.

Additionally, the model achieved a test accuracy of 90.74%, demonstrating its ability to generalize well to unseen data beyond the training and validation sets.

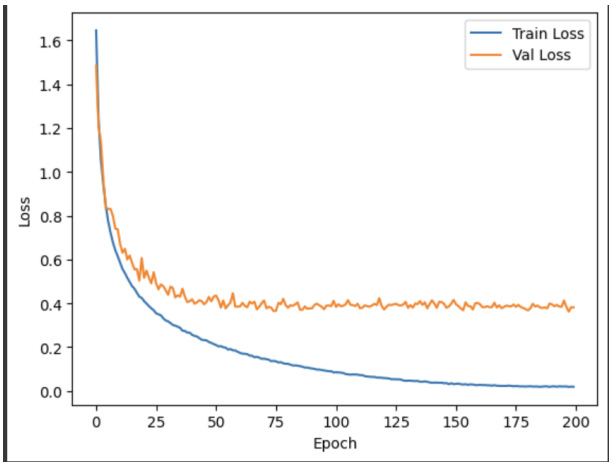

Figure 1: Training and Test loss

CITATIONS

- https://github.com/navoday01/ResNet5M-CIFAR10
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. "Deep Residual Learning for Image Recognition."
- https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831
- https://chat.openai.com/