Deep Learning Mini-Project

Team DLEnthusiasts

Sarang Kadakia[1], Vishwajeet Kulkarni[2]

New York University

[1]sk11634@nyu.edu [2]vk2630@nyu.edu

Abstract

Convolutional Neural Networks (CNNs) have a tendency to use a big number of layers in their network, but as there is a increase in the number of layers the problem of vanish- ing/exploding gradient arises. Therefore, increasing the test and training error rates. This issue can be combat-ted using ResNets. In this paper, we create a ResNet architecture where we achieve a test accuracy of 84.098%.

Overview

ResNets are deep neural networks that learn residual functions using skip connections. The key component in ResNet models is a residual block that implements:

$$ReLU\,(S(x) + F\,(x))$$

where $S(x)$ refers to the skipped connection and $F\,(x)$ is a block that implements *Input* → *Conv(3x3)* → *BN* → *ReLU* → *Conv(3x3)* → *BN* → *skip_connection* → *ReLU* → *Output*; here, "BN" stands for batch normalization. Chaining such blocks serially gives a deep ResNet.

Model parameters in such architectures include:

- B , the number of blocks in each block layer.
- C , the number of channels in the ith layer.
- F , the filter size in the ith layer
- K , the kernel size in the ith skip connection
- P , the pool size in the average pool layer

The implemented ResNet model consists of custom building blocks following the principles of BasicBlocks, each consisting of two convolutional layers with batch normalization and ReLU activation. The model is composed of multiple layers of these blocks, where each block layer maintains a consistent channel size. Our best-performing ResNet configuration for CIFAR-10 classification is as follows:

Final model architecture:

- **C**: [16,32,128,256]
- **F**: 3×3
- **K**: 1×1
- **P**: 4×4

The hyperparameters used to train this model:

- **Batch Size**: 16
- **Optimizer**: Adam
- **Learning Rate**: 0.00037167893563643987
- **Momentum**: 0.9

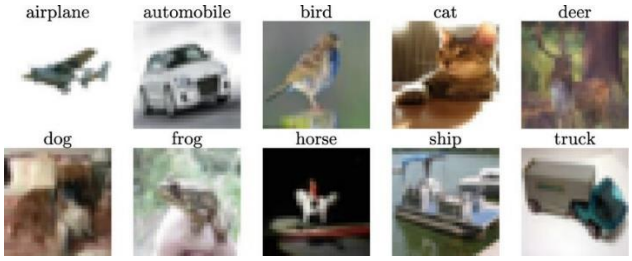
- **Weight Decay**: 0.0005
- **Annealing Cosine**: 50 cycles
- **Learning Rate Decay**: by 0.1

The techniques used to achieve this configuration, including Bayesian Optimization for hyperparameter selection and residual learning for efficient feature extraction, are discussed further in out Github repository (https://github.com/SARANG1018/DLEnthusiasts_Project).

Methodology

1. Dataset Processing

We used the CIFAR-10 dataset, consisting of 60,000 images across 10 classes, and partitioned it into training and testing subsets. Both the training and testing datasets were normalized using the standard CIFAR-10 mean and standard deviation values. We also employed dataloaders to efficiently batch-process the dataset during training.



2. Model Architecture

Our custom model, DLEnthusiasts_ResNet, is a variant of the ResNet architecture, designed with modular BasicBlock units and Squeeze-and-Excitation (SE) blocks for feature recalibration.

Key design variables include:

- N: The number of block layers.
- B: The number of blocks in each layer.
- C: The number of channels in each layer.
- F: The filter size in convolutional layers.
- K: The kernel size in skip connections.
- P: The pool size in the average pooling layer.

We experimented with different configurations of number of blocks, layer sizes, and channels, while ensuring the model remained within a 5 million parameter constraint.

3. Optimization Strategy

To determine the best hyperparameters, we leveraged Bayesian Optimization using Optuna. The parameters optimized include:

- Batch Size: {8, 16, 32}

- Learning Rate: Log-uniform search between 1e-5 to 1e-2
- Optimizer: {SGD, Adam}
- Momentum (for SGD): 0.9
- Weight Decay: 0.0005
- Filter Size, Kernel Size, and Pooling Size: tuned to find the best spatial transformations

We ran 20 trials of Bayesian Optimization, selecting the best configuration based on the lowest validation loss.

4. Training Process & Regularization

The model was trained using Stochastic Gradient Descent (SGD) with Cosine Annealing LR scheduler to dynamically adjust learning rates. We set:

- Epochs: 20
- Weight Decay: 5e-4 to prevent overfitting
- Gradient Accumulation: Used to stabilize updates when training with smaller batch sizes.

To mitigate overfitting, we used:

- Batch Normalization to stabilize activations
- Dropout layers in fully connected layers
- Squeeze-and-Excitation (SE) blocks to improve channel-wise feature importance.

5. Key Observations

- Hyperparameter Tuning Impact: Optimizing filter sizes, kernel sizes, learning rates, and batch sizes contributed significantly to accuracy improvements.
- SE Blocks Contribution: The inclusion of Squeeze-and-Excitation (SE) blocks improved feature recalibration and model performance.

6. Optimizers and Learning Rates (Based on Our Code Strategy)

We explored different optimizer strategies to improve model performance and stability. After multiple trials, we selected Adam as the primary optimizer due to its ability to efficiently handle adaptive learning rates and faster convergence on the CIFAR-10 dataset.

Adam (Adaptive Moment Estimation):

- Chosen as the final optimizer based on hyperparameter tuning.
- Used with learning rate = 0.0003716 (fine-tuned using Bayesian Optimization).
- Applied weight decay = 5e-4 to prevent overfitting and stabilize training.

Learning Rate Strategies

- Initial learning rate = 0.0003716, optimized through Bayesian Optimization.

- Applied cosine annealing scheduler with 200 max iterations for smooth learning rate adjustments.
- Implemented gradual learning rate decay to refine optimization over time.

These optimizations helped improve accuracy and model convergence while preventing overfitting. The final configuration achieved a balance between computational efficiency and accuracy improvement on the CIFAR-10 dataset.

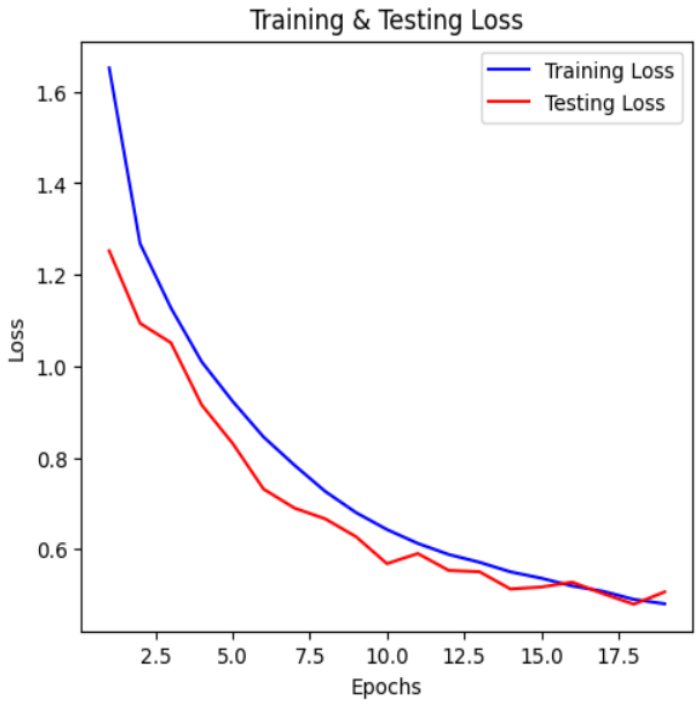
Results

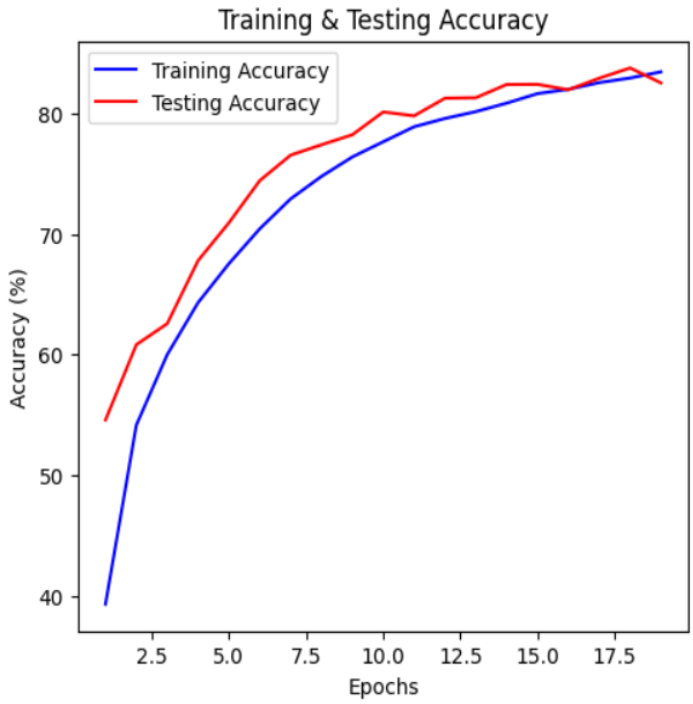
Our final model achieved a test accuracy of approximately 84% on the CIFAR-10 dataset. The best-performing model configuration used three block layers with channel sizes [16, 32, 128, 256].

The training and evaluation process was visualized using graphs that depict epoch-wise loss and accuracy trends. The following insights were observed:

- Training vs. Test Loss: The loss function steadily decreased over epochs, indicating that the model was learning effectively. However, there were minor fluctuations, suggesting possible areas for further optimization.
- Training vs. Test Accuracy: The accuracy improved progressively, stabilizing after a certain number of epochs. The model exhibited a general upward trend, confirming its capability to generalize well on unseen data.

The epoch vs. loss and epoch vs. accuracy plots clearly demonstrate that our model successfully learned the representations of CIFAR-10 classes, though there is room for further enhancement.





Conclusion

In this project, we built and trained a custom ResNet model from scratch to classify images in the CIFAR-10 dataset while ensuring parameter efficiency. Through Bayesian Optimization, we fine-tuned hyperparameters to maximize accuracy while maintaining computational feasibility. The use of SE blocks, optimized convolutional layers, and regularization techniques contributed to stabilizing and improving the model's performance. While we achieved a test accuracy of 84%, further refinements could enhance its effectiveness.

Future Scope

- **Enhanced Hyperparameter Tuning:** Use HyperOpt, Random Search, or Grid Search to explore better hyperparameters and implement adaptive learning rate schedules for improved efficiency.
- **Architectural Improvements:** Add more layers, alternative activation functions, and optimized batch normalization/dropout settings to enhance feature extraction and prevent overfitting.
- **Advanced Techniques:** Experiment with attention mechanisms, improved pooling strategies, and stronger data augmentation methods to boost model generalization and performance.

References

1] Github. 2021. Train CIFAR10 with PyTorch. https://github. com/kuangliu/pytorch-cifar. Accessed: 2024-04-12.

2] PyTorch Developers. 2024. CutMix and MixUp: PyTorch Documentation.