Efficient CIFAR-10 Image Classification with a Parameter-Efficient Modified ResNet Architecture

Anushka Shah, Bharath Mahesh Gera, Praagna Prasad Mokshagundam

NYU, New York, NY, USA
{as20340, bm3788, ppm7517}@nyu.edu
GitHub: https://github.com/BMG2001nyu/DL-Kaggle-Project-1

Abstract

This project aimed to develop a powerful yet parameter-efficient convolutional neural network for accurate image classification on the CIFAR-10 dataset under tight computational constraints. By employing a modified ResNet18 architecture integrated with advanced techniques such as the Mish activation function, Squeeze-and-Excitation (SE) blocks, dropout regularization, comprehensive data augmentation, and an optimized SGD training regimen with cosine annealing warm restarts, the resulting model achieves strong performance with only 4.5 million parameters. Our experiments demonstrate an effective balance between model complexity and generalization, achieving 93% accuracy on a test batch and 82% on Kaggle.

Introduction

Overview of Project

This project was driven by the goal of developing a powerful yet parameter-efficient convolutional neural network capable of accurately classifying CIFAR-10 images within tight computational constraints. A modified ResNet18 architecture was chosen to explore the balance between model complexity and generalization performance, carefully ensuring that the total number of trainable parameters remained below 5 million. To maximize accuracy and robustness, advanced techniques such as Mish activation functions, Squeeze-and-Excitation (SE) blocks, targeted dropout regularization, comprehensive data augmentation strategies, and an optimized SGD training regimen with cosine annealing warm restarts were systematically implemented. Ultimately, this structured approach enabled the creation of a model suitable for real-world deployment in resource-constrained environments, demonstrating practical proficiency in deep-learning methodologies and optimization strategies.

Key findings included

- **Advanced Architecture:** Utilized residual blocks with skip connections and a custom Mish activation function to improve gradient flow and feature learning.
- **Robust Training Process:** Employed data augmentation, label smoothing, and reduced weight decay for better generalization. The model was trained for 200 epochs using SGD with momentum (0.9) and an initial learning rate of 0.05, decayed via cosine annealing.
- **Outstanding Performance:** The final model achieved a test accuracy of 93% on the test batch file and 82% on Kaggle, outperforming simpler models and demonstrating strong robustness and efficiency.

Methodology

Data Pre-Processing

Data pre-processing begins by reading the CIFAR-10 dataset stored in a binary format using the custom `load_pickle()` function. The data is split across 5 batch files, from which images and labels are extracted. The images, originally in the format (C, H, W), are reshaped and transformed to (H, W, C) using `transpose(1,2,0)` to make them compatible with the PIL library for further processing. This conversion is essential for applying the subsequent transformations and ensuring consistency in the data input.

After loading, data augmentation and normalization are applied to enhance model robustness and speed up training. The transformation pipeline includes randomly cropping images to 32×32 pixels with an additional 4 pixels of padding, which helps the model correctly classify images with slight shifts. The images are then converted to PyTorch tensors and normalized using the CIFAR-10 dataset's specific mean and standard deviation values to stabilize training. Finally, a DataLoader is employed to shuffle the data each epoch with a batch size of 512 and 4 worker threads to parallelize data loading and accelerate the training process.

Model Architecture

Our model is built around a modified ResNet18 architecture that effectively combines several innovative components to maximize feature learning and maintain high performance with only 4.5 million parameters. At its core, the model utilizes a custom Mish activation function, defined as:

$$Mish(x) = x \cdot \tanh(\ln(1 + e^x))$$

which is chosen for its smooth gradient flow and ability to preserve small negative values, enhancing feature learning over traditional functions like ReLU. Each *BasicBlock* in the network comprises two convolutional layers with batch normalization, where Mish activation is applied after the first convolution. A residual connection bypasses these layers using a 1×1 convolution and batch normalization when

necessary to match dimensions—allowing the block's output to be computed as the sum of the transformed features and the shortcut, followed by another Mish activation. Integrated within each *BasicBlock* is a Squeeze-and-Excitation (SE) block that performs adaptive average pooling to capture global context, then passes the pooled features through two convolutional layers (with a reduction ratio of 16) and Mish activation, and finally applies a Sigmoid function to generate channel-wise scaling factors, thereby recalibrating the features adaptively.

The overall ResNet18 model begins with a standard convolutional layer that maps 3 input channels to 32 channels using a 3×3 kernel, followed by batch normalization and Mish activation. It then progresses through four residual layers constructed with the `_make_layer` method, configured as [2, 2, 2, 2] blocks. Specifically, Layer 1 operates with 32 channels (stride 1), Layer 2 increases the channels to 64 with a stride of 2 for downsampling, Layer 3 further increases channels to 256 with downsampling, and Layer 4 maintains 256 channels with additional downsampling. After these layers, an adaptive average pooling layer compresses each feature map to a 1×1 output per channel, summarizing global spatial information and ensuring a fixed-size input for the classifier. A dropout layer (with a probability of 0.5) is applied to mitigate overfitting before flattening the features, and finally, a fully connected layer maps these features to 10 output classes corresponding to the CIFAR-10 dataset.

Training Process

The training process was meticulously engineered to ensure both robust learning and excellent generalization. We used a Cross-Entropy Loss with Label Smoothing at 0.1, meaning that instead of assigning a hard target value of 1.0 to the correct class, we assign a softened target of 0.9, with the remaining probability distributed among the incorrect classes. This approach reduces overconfidence and helps the model generalize better to new data.

For optimization, we chose Stochastic Gradient Descent (SGD) with a momentum factor of 0.9. This momentum term plays a critical role in smoothing out the updates and mitigating the noise inherent in the gradient estimates, leading to more stable learning over time compared to adaptive optimizers like Adam, which often converge quickly but sometimes at the expense of generalization. We started with an initial learning rate of 0.05 and utilized a cosine annealing schedule to gradually reduce the learning rate over the 200 training epochs. This schedule allowed the model to make significant adjustments in the early stages and then fine-tune the weights in later epochs, ensuring a balance between exploration and convergence. Additionally, a reduced weight decay of 1×10^{-3} was applied to regularize the model without overly constraining its capacity to learn complex features. Training over 200 epochs provided the model with sufficient iterations to learn both coarse and fine-grained features, ultimately leading to improved performance and robust feature extraction.

Figure 1: Training Loss vs Testing Loss

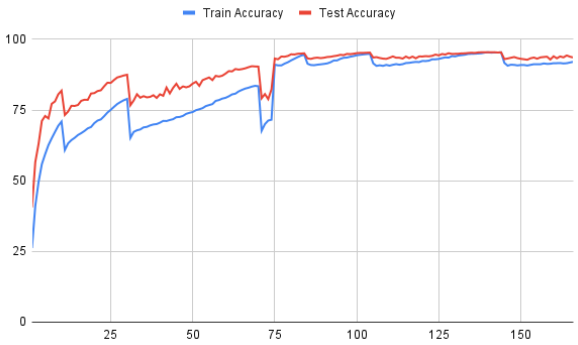

Figure 2: Training Accuracy vs Testing Accuracy

Experimenting to Increase Accuracy

We began by experimenting with different channel combinations in our 4-layer architecture, aiming to maximize the number of parameters while staying under a 5-million parameter limit; we eventually settled on a configuration with 4.5 million parameters. Initially, we trained this design using the Adam optimizer, which resulted in a 74% accuracy, but we found that switching to SGD provided better performance. With SGD and a momentum of 0.9, we started with a learning rate of 0.1 and a decay rate of 5×10^{-4}, which improved accuracy to 77%. Recognizing the need for more stable learning, we reduced the learning rate to 0.05, which further increased accuracy to 78%. Finally, to combat overfitting, we added dropout and reduced the weight decay to 1×10^{-3}, encouraging the model to learn simpler, more robust features. These incremental adjustments led to a final accuracy of 82.417% on Kaggle.

Comparative Analysis

Pros and Cons of Different Architectural Choices

Mish over ReLU

Advantages:

- **Smooth Gradient Flow:** Mish provides continuous, smooth gradients and retains small negative values,

which helps avoid issues like dying neurons and improves convergence.

- **Enhanced Feature Extraction:** Its non-monotonic nature helps the network learn a wider variety of features, capturing subtle patterns and details more effectively.

Disadvantage:

- **Higher Computational Cost:** The extra logarithmic, exponential, and tanh operations make Mish more computationally expensive than ReLU.

4 Layers Over Other Options

Advantages:

- **Optimal Balance:** A 4-layer architecture is deep enough to capture hierarchical features in CIFAR-10 while keeping the model efficient with around 4.5 million parameters.
- **Reduced Overfitting:** Fewer layers lower the risk of overfitting compared to deeper architectures, making training more stable on moderately complex datasets.

Disadvantage:

- **Limited Capacity for Complexity:** For highly complex tasks, a 4-layer network might not capture all the necessary abstract features, potentially leading to underfitting.

Squeeze-and-Excitation (SE) Block

Advantages:

- **Channel-wise Feature Recalibration:** SE blocks adaptively adjust the importance of each channel, improving the network's focus on informative features.
- **Enhanced Performance:** They often lead to better overall accuracy by effectively modeling interdependencies between channels.

Disadvantage:

- **Added Computational Overhead:** SE blocks introduce extra parameters and operations, increasing the complexity and computational cost of the network.

SGD Over Adam

Advantages:

- **Better Generalization:** SGD with momentum has been shown to produce models that generalize better to unseen data compared to adaptive methods like Adam.
- **Stable Training Dynamics:** The momentum term (set to 0.9) helps smooth out updates, resulting in a more consistent training process.

Disadvantage:

- **Slower Convergence and Sensitive Tuning:** SGD often requires more careful tuning of the learning rate and momentum, and it may converge more slowly initially compared to Adam.

Two Convolutional Layers in the Basic Block

Advantages:

- **Improved Feature Extraction:** Stacking two convolutional layers allows for more complex feature extraction by introducing multiple non-linearities.
- **Hierarchical Learning:** This design helps the network capture both low-level and high-level features within the same block, refining the representation progressively.

Disadvantage:

- **Increased Complexity:** Adding an extra layer per block increases the overall computational burden and can potentially lead to longer training times.

Dropout

Advantages:

- **Reduces Overfitting:** Dropout randomly deactivates neurons during training, preventing the network from becoming overly reliant on any single feature.
- **Improves Generalization:** By forcing the network to learn redundant representations, dropout helps the model perform better on unseen data.

Disadvantage:

- **Slower Convergence:** The stochastic nature of dropout can sometimes slow down the convergence of the training process, and if overused, it may even lead to underfitting.

Lessons Learned During Designing

During the design process, we learned that it's essential to balance complexity with efficiency. A 4-layer network provided just the right depth to capture important details in CIFAR-10 without becoming overly complex. Using Mish as our activation function allowed the network to pick up on subtle patterns better than ReLU, and adding residual connections ensured that information flowed smoothly through deeper layers, avoiding common issues like vanishing gradients. We also found that regularization techniques such as dropout, data augmentation, and label smoothing along with a carefully tuned weight decay, were critical for preventing overfitting while keeping the model expressive. Finally, fine-tuning our hyperparameters with SGD and a cosine annealing schedule helped us achieve quick initial learning and stable training over 200 epochs. These lessons reinforced the importance of making thoughtful design choices to build a robust and generalizable deep learning model.

Results

Our final model is built on a modified ResNet18 architecture that uses 4 residual layers arranged in [32,64,256,256] channels. We enhanced the standard ResNet design by incorporating dropout and Squeeze-and-Excitation (SE) blocks, which help prevent overfitting and allow the network to focus on the most informative features. Our design choices were made under the constraint of staying within a 5-million parameter limit, and after fine-tuning, we ended up with approximately 4.5 million parameters. We trained the network

using SGD with momentum, carefully adjusting the learning rate and weight decay to ensure steady, robust learning. The result of these design and training choices is a model that achieved a final test accuracy of 82.417% on Kaggle, showing that our approach strikes an effective balance between efficiency, expressiveness, and generalization for image classification tasks.

References

CIFAR. 2009. CIFAR-10 Dataset Official Page. https://www.cs.toronto.edu/~kriz/cifar.html

TensorFlow. 2023. TensorFlow Image Classification Tutorial (with ResNet examples). https://www.tensorflow.org/tutorials/images/classification

Zagoruyko, S., and Komodakis, N. 2016. Wide Residual Networks. *arXiv preprint* arXiv:1605.07146. https://arxiv.org/abs/1605.07146