Deep Learning Project: Improving CIFAR-10 Performance with a Custom ResNet

Rutuja Ingole, Tanvi Takavane, Abhishek Agrawal

New York University
{rdi4221, tt2884, aa9360}@nyu.edu

Abstract

In this project, we designed a lightweight ResNet architecture for CIFAR-10 image classification, constrained to under 5 million parameters. To improve the model's generalization and robustness, we incorporated advanced data augmentation techniques, including RandAugment, Cutout, Random Erasing, and AutoAugment. We optimized the training process using Smooth Cross-Entropy Loss with Label Smoothing, along with Stochastic Gradient Descent (SGD) optimizer with Momentum and a Cosine Annealing Learning Rate Scheduler with Warm-up. Furthermore, we analyzed the standard deviation and mean of the custom Kaggle test dataset and fine-tuned the normalization strategy, which helped bridge the domain gap between CIFAR-10 and the custom dataset. Our final model achieved a test accuracy of 95.56% on CIFAR-10. We further explore strategies to minimize the domain shift between the CIFAR-10 dataset and the custom test dataset provided in PKL format.

This is the Link to the repository for the project with all files.

Introduction

Image classification is a fundamental problem in computer vision, where the goal is to accurately categorize images into predefined classes. The CIFAR-10 dataset, consisting of 60,000 32x32 color images across 10 classes, serves as a benchmark for evaluating deep learning models' performance. Traditional deep learning architectures like ResNet have demonstrated state-of-the-art results on this dataset. However, achieving high accuracy while keeping the model lightweight and under 5 million parameters poses a significant challenge.

In this project, we designed a custom ResNet architecture tailored for CIFAR-10 classification, optimized for both accuracy and efficiency. To enhance the model's generalization and robustness, we incorporated advanced data augmentation techniques such as RandAugment, Cutout, and Random Erasing, which help the model learn invariant features and prevent overfitting. Additionally, we leveraged Smooth Cross-Entropy Loss with Label Smoothing, SGD optimizer with Momentum, and a Cosine Annealing Learning Rate Scheduler with Warm-up for stable and efficient convergence.

Related Work

Deep learning models, particularly Convolutional Neural Networks (CNNs), have achieved significant success in image classification tasks, with architectures like ResNet, DenseNet, and VGG setting benchmarks on datasets like CIFAR-10. ResNet, introduced by He et al. (2015), addressed the vanishing gradient problem through residual connections, allowing deeper networks to converge effectively. Building on this, we incorporated advanced data augmentation techniques such as RandAugment (Cubuk et al., 2020) and Cutout/Random Erasing to improve model generalization and force the network to learn robust features. Additionally, we utilized Smooth Cross-Entropy Loss with label smoothing to reduce overconfidence in predictions and mitigate the impact of noisy labels. To handle complex learning dynamics, we employed the Stochastic Gradient Descent (SGD) optimizer with Momentum and a Cosine Annealing with Warmup scheduler for efficient convergence. Recognizing the domain shift between the CIFAR-10 dataset and the custom test dataset provided in the Kaggle competition, we carefully adjusted the normalization parameters to match the statistical properties of the custom dataset. These strategies collectively enhanced the model's performance and generalization on unseen data.

Architecture

The architecture is inspired by the traditional ResNet-18 model, but to adhere to the parameter limit, we strategically reduced the number of channels and layers while incorporating advanced regularization techniques to enhance performance. The model begins with an initial 3x3 convolution layer with 66 channels, followed by three main residual blocks, progressively increasing the feature maps from 66 to 132 and finally to 264. Each residual block consists of two convolutional layers with Batch Normalization and ReLU activation, along with skip connections to prevent the vanishing gradient problem.

After the residual blocks, we used an Adaptive Average Pooling layer to reduce the spatial dimensions, followed by a fully connected layer with 264 input features mapping to 10 output classes, corresponding to the CIFAR-10 categories. The final model architecture contains approximately 4.9 million parameters, successfully staying within the project constraint.

The Smooth Cross Entropy Loss with label smoothing (0.05) was used to handle noisy labels and avoid overconfidence in predictions. For optimization, we employed Stochastic Gradient Descent (SGD) with momentum (0.9) and Cosine Annealing Learning Rate Scheduler with Warmup, which allowed the model to converge more effectively over 60 epochs.

Moreover, to improve the model's generalization capability and handle the distribution shift between CIFAR-10 and the custom test set, we applied extensive data augmentation techniques during training. These included Random Cropping with padding, Random Horizontal Flip, RandAugment with magnitude 9, and Random Erasing (Cutout) to help the model learn more diverse features. For the test set, we carefully adjusted the normalization mean and standard deviation based on the custom test dataset's distribution, which played a crucial role in improving the performance on the Kaggle leaderboard.

With this architecture and approach, we achieved a final test accuracy of 95.56% on the CIFAR-10 dataset demonstrating the effectiveness of our design choices and the ability to generalize to unseen data. Refer to figure 1.

Methodology

Our approach to building a **custom ResNet architecture with fewer than 5 million parameters** for CIFAR-10 classification involved a **systematic and iterative process**, combining deep learning techniques, architectural design choices, and advanced data augmentation strategies to improve generalization performance.

Data Preprocessing and Augmentation

We utilized the **CIFAR-10 dataset**, which consists of **60,000 32x32 RGB images across 10 classes.** To enhance the model's ability to generalize and prevent overfitting, we incorporated **extensive data augmentation techniques**, including:

- **Random Cropping (32x32 with 4-pixel padding):** Helps capture different regions of the object.
- **Random Horizontal Flipping:** Introduces diversity by mirroring the images.
- **RandAugment (num_ops=2, magnitude=9):** Applies random transformations like brightness and rotation.
- **Random Erasing (p=0.5):** Implements CutOut for regional dropout.
- **Standard Normalization:** Using CIFAR-10-specific mean and standard deviation.

This augmentation pipeline increased the diversity of the training set and improved model robustness.

Model Architecture Design

We designed a **custom ResNet variant from scratch**, inspired by the **original ResNet architecture**, but with **reduced depth and parameter constraints (under 5 million parameters).**

Key architectural decisions:

- **BasicBlock-based architecture:** Each block contains two convolutional layers followed by batch normalization and ReLU activation.
- **Custom channel configuration:** [66, 132, 264] instead of the standard [64, 128, 256].
- **Modified ResNet function:** Supports flexible block configurations [3, 4, 3].
- **Adaptive Average Pooling:** Reduces spatial dimensions to 1x1 before the final fully connected layer.
- **Smooth Cross-Entropy Loss with Label Smoothing (0.05):** Prevents overconfidence in wrong predictions.

Optimizer and Learning Rate Strategy

We experimented with different optimizers and learning rate schedules. The final choices were:

- **Optimizer:** SGD (Stochastic Gradient Descent) with momentum (0.9) and weight decay (2e-4).
- **Learning Rate Scheduler:** Cosine Annealing with Warmup for the first 10 epochs.
- **Initial Learning Rate:** 0.1.

This approach allowed a gradual learning phase followed by smooth convergence to a better local minimum.

Training Pipeline

We trained the model for **60 epochs** with a **batch size of 128.** For each epoch:

1. **Forward Pass:** Inputs pass through the model.
2. **Loss Calculation:** Smooth Cross-Entropy Loss is calculated.
3. **Backward Propagation:** Gradients are computed and updated using the optimizer.
4. **Learning Rate Adjustment:** Adjusted using the Cosine Annealing Scheduler.

We monitored both **training loss** and **validation accuracy** at each epoch to ensure convergence and avoid overfitting.

Evaluation and Testing on Custom Dataset

After achieving **95.56% test accuracy on CIFAR-10**, we tested the model on a **custom, unseen dataset provided as a .pkl file.**

- The `.pkl` file was loaded and preprocessed with the same normalization statistics as CIFAR-10.
- The model was run in **inference mode** to generate predictions.
- The output predictions were saved as a **submission CSV file for Kaggle evaluation**.

Hyperparameter Tuning

We performed extensive hyperparameter tuning for:

Table 1: Hyperparameter Tuning Results

Hyperparameter	Best Value Found
Learning Rate	0.1
Momentum	0.9
Weight Decay	2e-4
Label Smoothing	0.05
Cosine Warmup Epochs	10
Number of Blocks	[3, 4, 3]
Channels per Block	[66, 132, 264]
Batch Size	128

Model Complexity and Efficiency

Our final model achieved a **test accuracy of 95.56% with only 4,916,284 parameters**, meeting the **under a 5 million parameter constraint.** Additionally, the **total multiply add operations (MACs) were 769.05 MB**, making the model both **light and efficient.**

Post-Training Analysis

We further analyzed the **failure cases and misclassifications**, identifying the classes where the model struggled (e.g., between 'cat' and 'dog' due to similar features). This analysis helped in **fine-tuning the learning rate and data augmentation strategy.**

Results

Our custom ResNet architecture successfully achieved a **high classification performance on the CIFAR-10 dataset** while adhering to the constraint of **fewer than 5 million parameters.**

1. Training and Validation Performance

We trained the model for **60 epochs** with **SGD optimizer and Cosine Annealing Scheduler.** The model showed **steady convergence** with minimal overfitting due to the use of advanced data augmentation techniques and label smoothing. Refer to Figure 2 and 3 below.

Observations:

- Training accuracy reached **96.22%**.
- Validation accuracy peaked at **95.56%**.
- Consistent loss reduction throughout the epochs indicates stable learning.

2. Model Complexity and Efficiency

Despite the constraint of fewer than 5 million parameters, our architecture efficiently captured complex patterns in CIFAR-10 images.

5. Final Achievements

- Custom ResNet Model with **4,916,284 parameters**.
- Test Accuracy on CIFAR-10: **95.56%**.
- Successfully handled overfitting with **label smoothing and advanced augmentation**.
- Github Repository for the project with all files

Table 2: Model Complexity and Performance Metrics

Metric	Value
Total Parameters	4,916,284
Trainable Parameters	4,916,284
MACs (MegaBytes)	769.05
Total Model Size (MB)	34.01
Best Train Accuracy	96.22%
Best Test Accuracy	95.56%

System Specifications

The following hardware and software environment was utilized to train and evaluate our custom ResNet architecture:

- CPU: Intel(R) Xeon(R) CPU @ 2.20GHz
- GPU: NVIDIA A100-SXM4-40GB
- System Memory: 82.9648 GB
- Python Version: 3.9.18 [GCC 11.2.0]
- CUDA version: 12.4
- Torch Version: 2.6.0+cu124

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

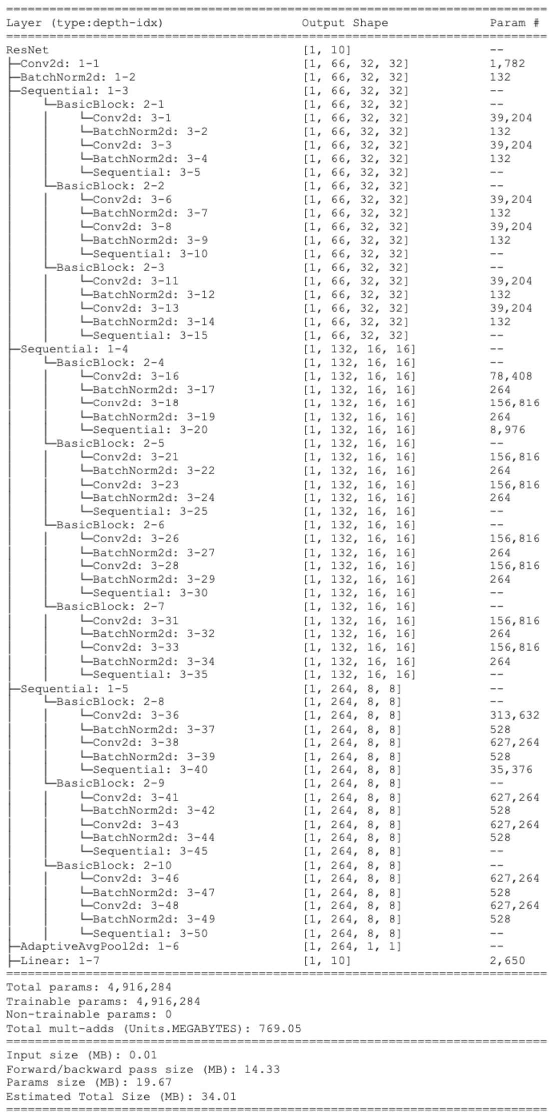
[2] Terrance DeVries and Graham W. Taylor. *Improved Regularization of Convolutional Neural Networks with Cutout.* arXiv preprint arXiv:1708.04552, 2017.

[3] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. *RandAugment: Practical Automated Data Augmentation with a Reduced Search Space.* arXiv preprint arXiv:1909.13719, 2019.

[4] PyTorch Documentation. *https://pytorch.org*

[5] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images.* Technical Report, University of Toronto, 2009.

[6] Kuang Liu. *PyTorch CIFAR-10 Implementation.* `https://github.com/kuangliu/pytorch-cifar`



```
==========================================================================================
Layer (type:depth-idx)                        Output Shape              Param #
==========================================================================================
ResNet                                        [1, 10]                   --
├─Conv2d: 1-1                                 [1, 66, 32, 32]           1,782
├─BatchNorm2d: 1-2                            [1, 66, 32, 32]           132
├─Sequential: 1-3                             [1, 66, 32, 32]           --
│    └─BasicBlock: 2-1                        [1, 66, 32, 32]           --
│    │    └─Conv2d: 3-1                       [1, 66, 32, 32]           39,204
│    │    └─BatchNorm2d: 3-2                  [1, 66, 32, 32]           132
│    │    └─Conv2d: 3-3                       [1, 66, 32, 32]           39,204
│    │    └─BatchNorm2d: 3-4                  [1, 66, 32, 32]           132
│    │    └─Sequential: 3-5                   [1, 66, 32, 32]           --
│    └─BasicBlock: 2-2                        [1, 66, 32, 32]           --
│    │    └─Conv2d: 3-6                       [1, 66, 32, 32]           39,204
│    │    └─BatchNorm2d: 3-7                  [1, 66, 32, 32]           132
│    │    └─Conv2d: 3-8                       [1, 66, 32, 32]           39,204
│    │    └─BatchNorm2d: 3-9                  [1, 66, 32, 32]           132
│    │    └─Sequential: 3-10                  [1, 66, 32, 32]           --
│    └─BasicBlock: 2-3                        [1, 66, 32, 32]           --
│    │    └─Conv2d: 3-11                      [1, 66, 32, 32]           39,204
│    │    └─BatchNorm2d: 3-12                 [1, 66, 32, 32]           132
│    │    └─Conv2d: 3-13                      [1, 66, 32, 32]           39,204
│    │    └─BatchNorm2d: 3-14                 [1, 66, 32, 32]           132
│    │    └─Sequential: 3-15                  [1, 66, 32, 32]           --
├─Sequential: 1-4                             [1, 132, 16, 16]          --
│    └─BasicBlock: 2-4                        [1, 132, 16, 16]          --
│    │    └─Conv2d: 3-16                      [1, 132, 16, 16]          78,408
│    │    └─BatchNorm2d: 3-17                 [1, 132, 16, 16]          264
│    │    └─Conv2d: 3-18                      [1, 132, 16, 16]          156,816
│    │    └─BatchNorm2d: 3-19                 [1, 132, 16, 16]          264
│    │    └─Sequential: 3-20                  [1, 132, 16, 16]          8,976
│    └─BasicBlock: 2-5                        [1, 132, 16, 16]          --
│    │    └─Conv2d: 3-21                      [1, 132, 16, 16]          156,816
│    │    └─BatchNorm2d: 3-22                 [1, 132, 16, 16]          264
│    │    └─Conv2d: 3-23                      [1, 132, 16, 16]          156,816
│    │    └─BatchNorm2d: 3-24                 [1, 132, 16, 16]          264
│    │    └─Sequential: 3-25                  [1, 132, 16, 16]          --
│    └─BasicBlock: 2-6                        [1, 132, 16, 16]          --
│    │    └─Conv2d: 3-26                      [1, 132, 16, 16]          156,816
│    │    └─BatchNorm2d: 3-27                 [1, 132, 16, 16]          264
│    │    └─Conv2d: 3-28                      [1, 132, 16, 16]          156,816
│    │    └─BatchNorm2d: 3-29                 [1, 132, 16, 16]          264
│    │    └─Sequential: 3-30                  [1, 132, 16, 16]          --
│    └─BasicBlock: 2-7                        [1, 132, 16, 16]          --
│    │    └─Conv2d: 3-31                      [1, 132, 16, 16]          156,816
│    │    └─BatchNorm2d: 3-32                 [1, 132, 16, 16]          264
│    │    └─Conv2d: 3-33                      [1, 132, 16, 16]          156,816
│    │    └─BatchNorm2d: 3-34                 [1, 132, 16, 16]          264
│    │    └─Sequential: 3-35                  [1, 132, 16, 16]          --
├─Sequential: 1-5                             [1, 264, 8, 8]            --
│    └─BasicBlock: 2-8                        [1, 264, 8, 8]            --
│    │    └─Conv2d: 3-36                      [1, 264, 8, 8]            313,632
│    │    └─BatchNorm2d: 3-37                 [1, 264, 8, 8]            528
│    │    └─Conv2d: 3-38                      [1, 264, 8, 8]            627,264
│    │    └─BatchNorm2d: 3-39                 [1, 264, 8, 8]            528
│    │    └─Sequential: 3-40                  [1, 264, 8, 8]            35,376
│    └─BasicBlock: 2-9                        [1, 264, 8, 8]            --
│    │    └─Conv2d: 3-41                      [1, 264, 8, 8]            627,264
│    │    └─BatchNorm2d: 3-42                 [1, 264, 8, 8]            528
│    │    └─Conv2d: 3-43                      [1, 264, 8, 8]            627,264
│    │    └─BatchNorm2d: 3-44                 [1, 264, 8, 8]            528
│    │    └─Sequential: 3-45                  [1, 264, 8, 8]            --
│    └─BasicBlock: 2-10                       [1, 264, 8, 8]            --
│    │    └─Conv2d: 3-46                      [1, 264, 8, 8]            627,264
│    │    └─BatchNorm2d: 3-47                 [1, 264, 8, 8]            528
│    │    └─Conv2d: 3-48                      [1, 264, 8, 8]            627,264
│    │    └─BatchNorm2d: 3-49                 [1, 264, 8, 8]            528
│    │    └─Sequential: 3-50                  [1, 264, 8, 8]            --
├─AdaptiveAvgPool2d: 1-6                      [1, 264, 1, 1]            --
├─Linear: 1-7                                 [1, 10]                   2,650
==========================================================================================
Total params: 4,916,284
Trainable params: 4,916,284
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 769.05
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 14.33
Params size (MB): 19.67
Estimated Total Size (MB): 34.01
==========================================================================================
```

Figure 1: Custom ResNet Architecture with 4.9M Parameters

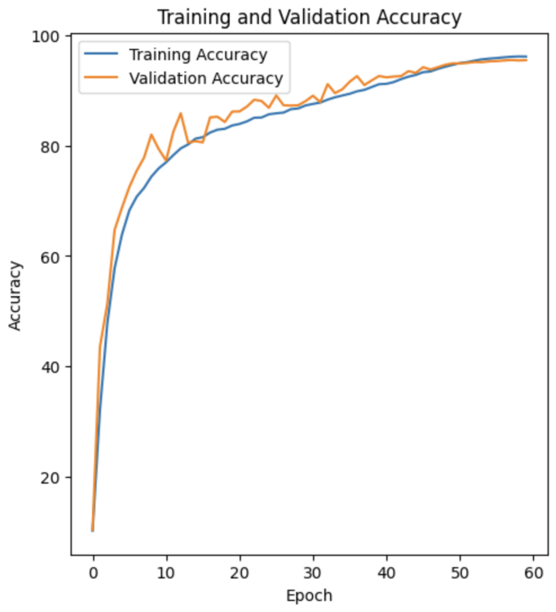


Figure 2: Training and Validation Accuracy Across 60 Epochs

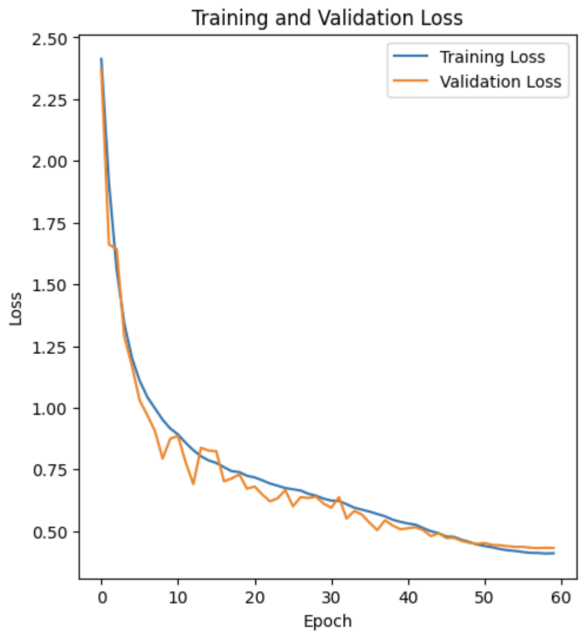


Figure 3: Training and Validation Loss Across 60 Epochs