Optimizing ResNet Architecture for CIFAR-10 under Parameter Constraints

Ranjan Patil, Nishanth Kotla , Navdeep Mugathihalli Kumaregowda
New York University
sp8171@nyu.edu, nk3968@nyu.edu, nm4686@nyu.edu
GitHub Repository: `https://github.com/ranjan2601/ResNet-MiniProject`

Abstract

We propose a compact ResNet-based model tailored for CIFAR-10 image classification under a strict five-million-parameter limit. Employing Cosine Annealing for smoother learning rate decay, CutMix augmentation for enhanced robustness, and Test-Time Augmentation (TTA) for improved inference, our approach achieves competitive accuracy without inflating model size. The resulting *ResNetSmall* architecture underscores how thoughtful architectural scaling and advanced training strategies can deliver strong performance in resource-constrained settings. Our findings demonstrate that sub-five-million-parameter networks can match or exceed larger counterparts, making them suitable for edge devices with limited computational resources. Moreover, the synergy of targeted regularization and well-chosen hyperparameters highlights the potential of optimizing classical ResNet designs for modern, efficiency-driven applications. In the Kaggle competition *Deep Learning Spring 2025: CIFAR 10 Classification*, our team **NeuralVibes** achieved a test score of **0.86113**, further validating the effectiveness of our approach.

Introduction

Residual Networks (ResNet) have transformed image classification by introducing skip connections that alleviate vanishing gradient issues, enabling deeper and more robust convolutional neural networks. In this project, we adapt and refine ResNet specifically for the CIFAR-10 dataset while respecting a strict cap of under 5 million trainable parameters. Central to this effort is our custom *ResNetSmall* architecture, inspired by the original ResNet design but carefully scaled to balance depth and width. Verified through `torchinfo`, the final model comprises approximately 4.74 million parameters. This parameter-efficient design ensures deployment feasibility on resource-constrained platforms without sacrificing classification accuracy.

This project consists of pipeline with multiple steps, from network construction to training and optimization. We begin by performing a systematic grid search over key hyperparameters—such as learning rate, momentum, weight decay, and scheduler settings—to identify an effective training strategy. After comparing optimizers, we adopt Stochastic Gradient Descent (SGD) with momentum and weight decay as it consistently outperforms other approaches in both convergence speed and generalization. We further enhance performance through multiple techniques: (1) Cosine Annealing learning rate scheduling for smoother convergence, (2) Mixup augmentation to improve robustness and reduce overfitting, and (3) Test-Time Augmentation (TTA) to boost final predictive accuracy. Together, these methods yield a high-performing yet compact ResNet-based classifier, showcasing that careful architectural choices and well-tuned optimization can achieve strong results within stringent parameter budgets.

Methodology

Dataset

We utilized the CIFAR-10 dataset comprising 60,000 RGB images (32×32 pixels), split into 50,000 training and 10,000 test images across 10 classes.

Model Architecture

We designed a compact *ResNetSmall* architecture inspired by ResNet's **BasicBlock** configuration (see Figure 1). The final model includes four layers with $[4, 4, 4, 3]$ blocks, keeping parameters below five million (precisely 4.74 million) as verified with `torchinfo`. Below, we outline the key components and design choices:

- **Residual Learning:** Each block learns a residual function $F(x)$, and the output is computed as:

  $$y = F(x) + x.$$

  This formulation helps mitigate vanishing gradients and accelerates training in deeper networks.
- **BasicBlock-based Architecture:** We use the standard *BasicBlock* from ResNet, which consists of two 3×3 convolutional layers, each followed by Batch Normalization (BN) and ReLU activation. The skip connection bypasses these convolutional layers to add the original input directly to the output.
- **Block Configuration $[4, 4, 4, 3]$:** Instead of employing large numbers of channels in each stage, we balance depth and width to respect the < 5 million parameter constraint. Each stage may include a stride of 2 in its first block for spatial downsampling and channel doubling:

1. *Stage 1*: Input channels are expanded from 3 (RGB) to an intermediate level (e.g., 32), repeated 4 times.
2. *Stage 2*: Channels double (e.g., 64), repeated 4 times.
3. *Stage 3*: Channels double again (e.g., 128), repeated 4 times.
4. *Stage 4*: Channels double once more (e.g., 256), repeated 3 times.

With each stage, the network gains representational power while controlling overall parameter growth.

- **Adaptive Average Pooling:** After the final convolutional block (output shape $\sim 256 \times 4 \times 4$), an `AdaptiveAvgPool2d` compresses each feature map to 1×1, producing a 256-dimensional feature vector per image. This removes the need for large fully connected layers.
- **Final Classification Layer:** We flatten the pooled feature maps and feed them into a single `Linear` layer (dimension $256 \to 10$) to produce logits for the 10 CIFAR-10 classes.
- **Skip Connections:** Each residual block includes a shortcut connection (skip) that directly adds the input to the block's output, ensuring smooth gradient flow and improved convergence, especially as depth increases.
- **Overall Parameter Efficiency:** Through careful tuning of the channel progression and number of blocks, our model maintains strong representational capacity while keeping the parameter count at approximately 4.74M, meeting the stringent < 5M requirement.

The model architecture is shown in Figure 1.

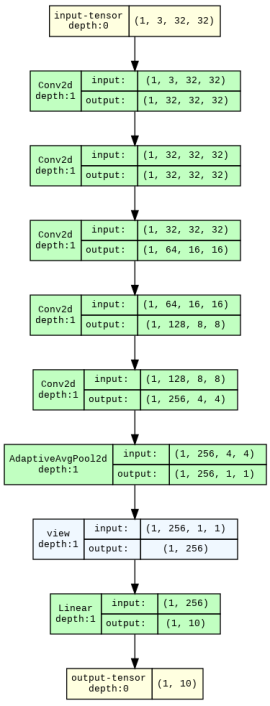

Figure 1: Architecture of the ResNetSmall model.

Training Approach

The initial training setup involved training the ResNetSmall model for 100 epochs with the following configuration:

- **Optimizer**: SGD (learning rate=0.1, momentum=0.9, weight decay=5e-4)
- **Scheduler**: MultiStepLR with milestones at epochs 50 and 75
- **Loss Function**: CrossEntropy
- **Data Augmentation**: Random cropping and horizontal flip

Optimization Techniques

To enhance the model's performance, we optimized it using an advanced configuration and trained it for 200 epochs. The updated setup included:

- **Optimizer**: SGD (learning rate=0.1, momentum=0.9, weight decay=5e-4)
- **Scheduler**: CosineAnnealingLR
- **Loss Function**: CrossEntropy
- **Data Augmentation**: CutMix, horizontal flip, Gaussian blur, and random erasing
- **Test-Time Augmentation (TTA)**: Applied to optimize performance on unseen data

The advantages of these optimization techniques are as follows:

- **CosineAnnealingLR**: Provides a smooth, gradual decay of the learning rate, enabling better convergence compared to the step-wise reduction of MultiStepLR, especially in later epochs.
- **CutMix Augmentation**: Enhances generalization by mixing cropped regions of images and their labels, reducing overfitting and improving robustness over standard augmentations like flip and crop.
- **Gaussian Blur**: Introduces variability by smoothing images, helping the model learn features resilient to noise and minor distortions.
- **Random Erasing**: Simulates occlusions, encouraging the model to focus on diverse image regions and further preventing overfitting.
- **Test-Time Augmentation (TTA)**: Boosts accuracy on unseen data by averaging predictions across multiple augmented versions of test images, leveraging the model's robustness to variations.

Metric	Initial Model	Optimized Model
Parameters	4.74M	4.74M
Optimizer	SGD	SGD
Scheduler	Multi Step LR	Cos. Anneal.
Epochs	100	200
Augmentation	Crop, Flip	CutMix, Blur, Flip, Erasing

Table 1: Comparison of initial and optimized ResNetSmall on CIFAR-10.

Results

We evaluated our ResNetSmall model on the CIFAR-10 dataset under two configurations: an initial model and an optimized model, with training details described in the above sections.

The initial model, trained for 100 epochs, achieved a test accuracy of 93.99%. The accuracy curves, loss curves, and confusion matrix generated on the CIFAR-10 test data are shown in Figures 2, 3, and 4, respectively.

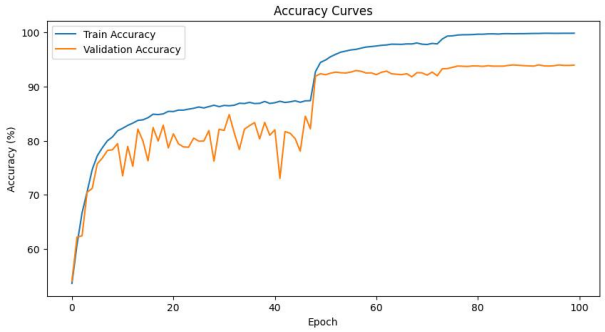

Figure 2: Accuracy curve of the initial ResNetSmall model.

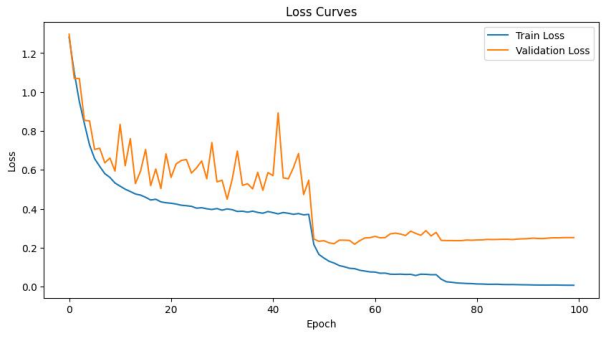

Figure 3: Loss curve of the initial ResNetSmall model.

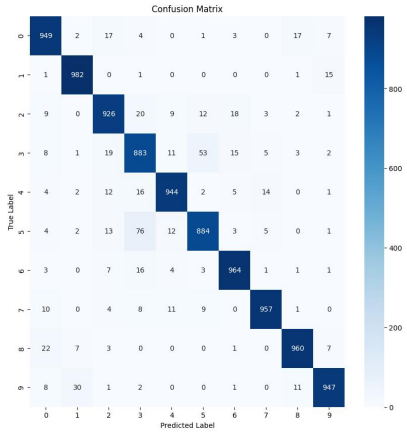

Figure 4: Confusion matrix of the initial ResNetSmall model on CIFAR-10 test data.

The optimized model, trained for 200 epochs with CutMix and additional augmentations, achieved a training accuracy of approximately 80–85% and a test accuracy of 90–95%. This inversion, where test accuracy exceeds training accuracy, is attributed to CutMix's strong regularization, which prevents overfitting by mixing training samples, and the use of Test-Time Augmentation (TTA), which enhances generalization on unseen data. The corresponding accuracy curves, loss curves, and confusion matrix are presented in Figures 5, 6, and 7, respectively.

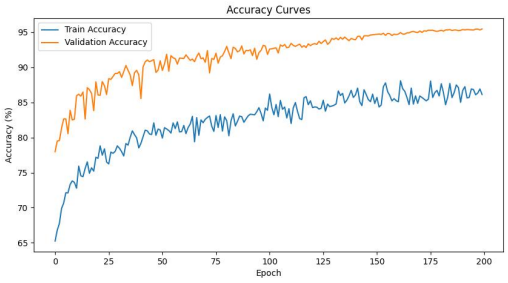

Figure 5: Accuracy curve of the optimized ResNetSmall model.

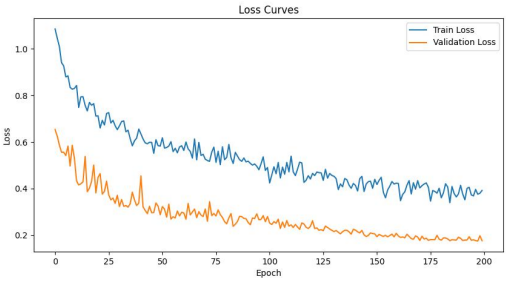

Figure 6: Loss curve of the optimized ResNetSmall model.

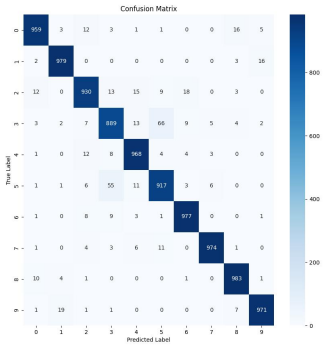

Figure 7: Confusion matrix of the optimized ResNetSmall model on CIFAR-10 test data.

Conclusion

Through careful experimentation with learning rate schedules and data augmentation, our ResNetSmall model achieved a training accuracy of 80–85% and a test accuracy of 90–95% on CIFAR-10, despite strict parameter limits. Trained for 200 epochs with CutMix and Test-Time Augmentation (TTA), the model demonstrated strong generalization, with its performance further confirmed by our participation in the Kaggle competition *Deep Learning Spring 2025: CIFAR 10 Classification*. In this challenge, our team **NeuralVibes** secured a test score of **0.86113**, underscoring the practical benefits of our parameter-efficient design and optimization strategies.

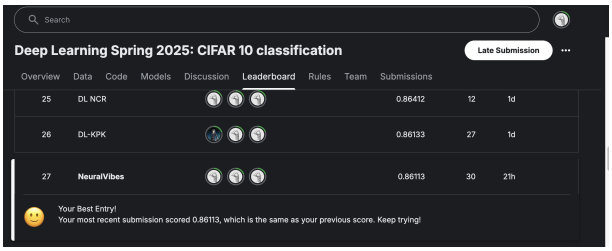

Figure 8: Screenshot of Kaggle submission score for the Deep Learning Spring 2025: CIFAR 10 Classification competition.

For future work, exploring additional regularization like MixUp or AutoAugment could further improve generalization. Testing on complex datasets like CIFAR-100 or ImageNet would assess scalability. Automated tools, such as Bayesian optimization, could enhance tuning efficiency. Lastly, interpretability methods like Grad-CAM could reveal how augmentation shapes feature learning, advancing transparent AI development.

Limitations and Challenges: During the development process, we encountered several challenges. In particular, hyperparameter tuning required a careful balance between learning rate decay and data augmentation strategies to prevent overfitting. Additionally, the strict parameter constraint (¡5M) limited our architectural flexibility, necessitating precise scaling of network depth and width. We addressed these issues through advanced regularization techniques such as CutMix and Test-Time Augmentation (TTA).

Reproducibility: To ensure transparency and reproducibility, all experimental logs, model summary outputs (via `torchinfo`), and detailed code execution results are available in our GitHub repository (link provided on the first page). This repository allows others to verify and build upon our work.

References

[1] `https://arxiv.org/pdf/2010.01412` – Deep Learning Optimization Techniques.

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. `https://arxiv.org/pdf/1512.03385`.

[3] Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-Excitation Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. `https://arxiv.org/pdf/1709.01507`.

[4] Foret, P., Kleiner, A., Mobahi, H., & Neyshabur, B. (2021). Sharpness-Aware Minimization for Efficiently Improving Generalization. In *ICLR*. `https://github.com/davda54/sam`.

[5] Moskomule. SAM PyTorch Implementation. `https://github.com/moskomule/sam.pytorch`.

[6] Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., & Yoo, Y. (2019). CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. *ICCV*. `https://arxiv.org/pdf/1905.04899`.

[7] Zhang, H., Cisse, M., Dauphin, Y.N., & Lopez-Paz, D. (2018). Mixup: Beyond Empirical Risk Minimization. *ICLR*. `https://arxiv.org/pdf/1710.09412`.

[8] Cubuk, E.D., Zoph, B., Shlens, J., & Le, Q.V. (2019). AutoAugment: Learning Augmentation Policies from Data. *CVPR*. `https://arxiv.org/pdf/1805.09509`.

[9] Müller, R., Kornblith, S., & Hinton, G.E. (2019). When Does Label Smoothing Help? *NeurIPS*. `https://arxiv.org/pdf/1906.02629`.

[10] Tan, M., & Le, Q.V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *ICML*. `https://arxiv.org/pdf/1905.11946`.

[11] Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. *arXiv preprint*. `https://arxiv.org/pdf/1503.02531`.