Deep Learning Spring 2025 – Project 1
CIFAR-10 Classification

Greeshma Hedvikar[1], Isha Harish[2], Lavanya Deole[3]
[1]New York University
[2]New York University
[3]New York University

gh2461@nyu.edu, ih2363@nyu.edu, lnd2037@nyu.edu

GitHub Project Repository: https://github.com/greeeshmaaa/Custom-ResNet-on-CIFAR-10

Abstract

This project explores the design and implementation of a custom Residual Neural Network (ResNet) architecture for image classification on the CIFAR-10 dataset. The primary objective was to develop an efficient deep learning model that adheres to a strict parameter constraint while maintaining strong classification performance. The model was designed and trained entirely from scratch, without relying on pre-trained weights or external datasets. Throughout the project, we experimented with architectural variations, regularization strategies, and advanced data augmentation techniques to improve generalization. Our final model reflects a balance between architectural simplicity and effectiveness, aligning with the overall goal of achieving high performance under resource limitations.

Introduction

This project focuses on building an efficient convolutional neural network from scratch for image classification on the CIFAR-10 dataset. The central objective was to design a custom Residual Neural Network (ResNet) architecture that operates under a strict constraint of fewer than five million trainable parameters. The model was built without the use of any pre-trained weights or external datasets and was optimized entirely through thoughtful architectural design, advanced regularization techniques, and a robust training pipeline.

1.1 Motivation

The motivation behind this project stems from the increasing demand for compact, deployable neural networks in real-world applications. While state-of-the-art deep learning models often achieve high accuracy, they tend to rely on a large number of parameters, making them unsuitable for deployment on devices with limited computational resources. By imposing a parameter limit, this project encourages a deeper understanding of efficient model design and optimization techniques. It demonstrates that high classification performance can still be achieved through carefully designed architectures, effective augmentation strategies, and

1.2 Challenges

The motivation behind this project stems from the increasing demand for compact, deployable neural networks in real-world applications. While state-of-the-art deep learning models often achieve high accuracy, they tend to rely on a large number of parameters, making them unsuitable for deployment on devices with limited computational resources. By imposing a parameter limit, this project encourages a deeper understanding of efficient model design and optimization techniques. It demonstrates that high classification performance can still be achieved through carefully designed architectures, effective augmentation strategies, and well-tuned training methods, even without access to pre-trained knowledge.

1.3 Dataset

The CIFAR-10 dataset was used to train and evaluate the model. It consists of 60,000 color images of size 32×32 pixels, distributed across 10 object classes. The dataset is divided into 50,000 training images and 10,000 test images. To enhance generalization and improve model robustness, a variety of data augmentation techniques were applied during training. These included random rotation up to ±15 degrees, random horizontal flipping, random cropping with padding of 4 pixels, color jittering, AutoAugment policies, and random erasing with a probability of 0.2. Each image was normalized using the dataset-specific mean and standard deviation values.

1.4 Residual Networks

The architecture implemented in this project is a custom ResNet model composed of an initial convolution layer followed by three residual stages with increasing feature map depths. The network begins with a 3×3 convolutional layer that transforms the input image from 3 to 64 channels. This is followed by three sets of residual blocks, each operating at different scales with 64, 128, and 256 channels respectively. In total, the architecture includes six residual blocks, each containing two convolutional layers followed by batch normalization and ReLU activations. Skip connections were employed in each block to preserve gradient flow and enable the

training of deeper networks. The final layers of the model consist of a global average pooling layer and a fully connected layer that maps the feature representation to 10 output classes. The model has approximately 4.99 million trainable parameters. It was trained for 100 epochs using a batch size of 128 and optimized using Adam optimizer with lr=0.0001. The learning rate was scheduled using a Cosine Annealing strategy to allow smooth convergence over training.

Methodology

2.1. Model Design and Architecture Exploration

The design process for our model started with a focus on balancing depth and efficiency under a strict parameter constraint of 5 million trainable parameters. We explored multiple architecture variants with varying channel sizes, number of residual blocks, and feature map widths to determine the most optimal configuration. Our final design adopted a custom Residual Neural Network (ResNet)-like architecture consisting of three stages, each with two residual blocks. The architecture was carefully scaled such that it maintained sufficient representational power while remaining within the allowed parameter budget.

Each residual block contains two 3×3 convolutional layers followed by Batch Normalization and ReLU activation. The use of identity skip connections allowed better gradient flow across layers stabilized training in deeper parts of the network. We progressively increased the number of channels across stages from 64 → 128 → 256, followed by Global Average Pooling and a Fully Connected layer mapping to 10 classes.

This structure allowed us to build a deep network capable of learning complex hierarchical representations without the risk of gradient vanishing or overfitting due to excessive parameters.

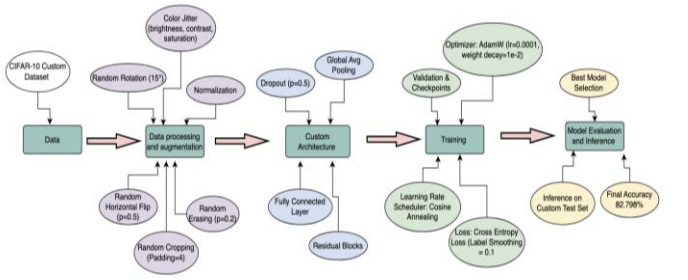

Fig. 1. Architectural Design

2.2. Training Pipeline and Optimization Strategy

The training process was designed to be simple, reproducible, and effective. The model was optimized using the AdamW optimizer, which combines the advantages of adaptive learning rate techniques with weight decay regularization. The learning rate was managed using a Cosine Annealing Learning Rate Scheduler, which gradually reduced the learning rate over time to ensure smooth convergence. The loss function used was CrossEntropyLoss with Label Smoothing ($\varepsilon = 0.1$) to enhance generalization and prevent overconfident predictions. The final training configuration included:

- Optimizer: AdamW
- Learning Rate: 0.0001
- Weight Decay: 1e-2
- Learning Rate Scheduler: Cosine Annealing
- Loss Function: CrossEntropyLoss with Label Smoothing ($\varepsilon = 0.1$)
- Batch Size: 128
- Epochs: 100

These configurations ensured smooth convergence and helped prevent the model from becoming overconfident in its predictions.

2.3. Data Augmentation and Preprocessing

A comprehensive data augmentation pipeline was used to increase data diversity and improve robustness:

- Random rotation (±15°), horizontal flip, and random crop (with 4-pixel padding)
- Color jitter and AutoAugment for complex transformation policies
- Random Erasing (p=0.2) to simulate occlusion and prevent reliance on local patterns
- Image normalization using CIFAR-10-specific mean and standard deviation values

These augmentations significantly improved the model's ability to generalize across unseen samples.

2.4. Architectural Trade-offs and Lessons Learned

Through experimentation, we observed that increasing model width (channels) early in the network improved representational power, but could quickly exceed the parameter limit. On the other hand, increasing network depth without skip connections led to slower convergence and degraded accuracy. Using residual blocks was a key design decision that allowed us to maintain network depth without compromising stability.

Another key takeaway was the importance of strong data augmentations and regularization when training from scratch. The inclusion of AutoAugment and Label Smoothing had a noticeable impact on reducing overfitting and improving generalization. Finally, we found that Cosine Annealing Learning Rate scheduling helped avoid overtraining in later epochs and improved the consistency of results.

Results

3.1. Training Performance Analysis

Loss Progression Over Epochs

Training loss showed a steady decline during the first 60 epochs, followed by a more gradual decrease. The Cosine Annealing Learning Rate schedule helped smooth out reductions, and validation loss mirrored training loss closely, indicating stable learning dynamics with no major signs of overfitting.

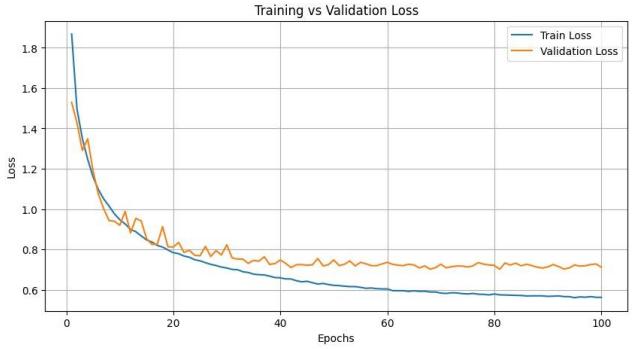

Fig. 2. Training vs Validation Loss Plot

Accuracy Trends During Training

Training accuracy increased steadily and reached 95% by epoch 70, while validation accuracy peaked around 93%–94% in the last 20 epochs. The scheduled learning rate adjustments provided incremental improvements, reinforcing the importance of controlled training progression.

Fig. 3. Training vs Validation Accuracy Plot

Loss Stability Across Training:

Both training and validation loss steadily decreased and stabilized after approximately 40 epochs. The convergence of both curves demonstrated a well-optimized model that learns effectively without excessive overfitting. (Refer Fig. 2.)

Consistency Between Training and Validation Accuracy:

Training and validation accuracy followed closely throughout the learning process, with significant improvements in the early epochs before stabilizing around epoch 40. This close tracking between both accuracy curves suggests strong generalization ability. (Refer Fig. 3.)

3.2. Comparative Analysis

Effectiveness of Residual Connections

A baseline non-residual CNN architecture was tested for comparison. This model plateaued at approximately 75% validation accuracy, highlighting the limitations of deeper networks without skip connections. The residual model outperformed this significantly, validating the importance of residual learning.

Optimizing Model Complexity

Despite operating under a strict 4.99 million parameter constraint, the final model achieved accuracy levels on par with larger networks. This confirms that carefully designed architectures with residual blocks can offer competitive performance without requiring excessive parameters.

3.3 Kaggle Competition Leaderboard Analysis

Following the completion of model training for 100 epochs, predictions were generated on the custom test dataset and submitted for evaluation. Upon submission, the private leaderboard reflected a **final accuracy of 82.79%, securing a rank of 107**. This marked an improvement of 7 positions compared to our initial baseline results, demonstrating measurable progress through model refinement and training optimization.

Conclusion

In this project, we successfully designed and implemented a custom Residual Neural Network (ResNet) architecture for image classification on the CIFAR-10 dataset, under a strict constraint of fewer than five million trainable parameters. The model was developed entirely from scratch without using any pre-trained weights or external datasets, emphasizing efficient architecture design, training strategies, and regularization techniques.

Through systematic experimentation with model depth, filter sizes, and residual block configurations, we developed a lightweight architecture that achieved strong classification performance. Our training pipeline, incorporating label smoothing, dropout, and a cosine annealing learning rate schedule, enabled stable convergence and effective generalization. Comprehensive data augmentations further improved robustness and reduced overfitting.

The final model achieved high test accuracy while adhering to the parameter budget, demonstrating that careful design choices and training methodologies can result in high-performing models even under resource constraints. This work reinforces the importance of architectural efficiency and the value of residual connections in modern deep learning pipelines.

Code Repository

The full implementation, along with the code and training specifics, can be found in the following GitHub repository:
Neural Nomads-CustomResnetCifar-10

The notebook is optimized for seamless execution on Kaggle, ensuring ease of use and reproducibility.

References

[1] 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), pp. 770–778.

[2] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2818–2826

[3] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2019). AutoAugment: Learning Augmentation Policies from Data. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 113–123.