# Modified ResNet Trained on CIFAR-10 Classification

**Team Name: teamname**
Chenke Wang
cw4565
Tianyu Liu
tl3191
Zhaochen Yang
zy2189

## Project codebase

The complete codebase for this project is available on GitHub:
`https://github.com/lingtouyangLeo/dl_prject_1`

## Abstract

Deep learning models, particularly Residual Networks (ResNets), play a crucial role in image classification. In this project, we design a modified ResNet for CIFAR-10, maintaining a strict parameter constraint of 5 million.

To optimize performance, we reduce convolutional layers per residual block, adjust channel sizes, and apply techniques like SGD with momentum, weight decay, and cosine annealing learning rate scheduling. Our model achieves 94.06% test accuracy with only 2.7M parameters, outperforming standard ResNet-18 (11.2M).

However, a deeper 4-layer ResNet overfits CIFAR-10 and generalizes poorly to new datasets (only 75% accuracy). This highlights the importance of compact architectures for small datasets. Future work includes enhancing generalization through knowledge distillation and model quantization.

## Introduction

Deep learning has revolutionized image classification tasks, and residual networks (ResNets) have played a pivotal role in achieving state-of-the-art performance. The primary objective of this project is to design a modified ResNet architecture that achieves high test accuracy on the CIFAR-10 dataset while maintaining a strict constraint of having no more than 5 million parameters.

Residual networks (ResNets) introduce shortcut connections that alleviate the vanishing gradient problem in deep networks. These skip connections allow the gradient to propagate more effectively, enabling the training of very deep architectures. Standard ResNet models, such as ResNet-18 and ResNet-34, serve as strong baselines for CIFAR-10 classification. However, their parameter count often exceeds the 5M constraint, necessitating modifications to the architecture.

In this project, we explore several architectural modifications to ResNet, including:

- Adjusting the number of channels in convolutional layers to reduce parameter count.
- Reducing the number of convolutional layers per residual block.
- Incorporating batch normalization and dropout to improve generalization.
- Implementing various data augmentation strategies to enhance model robustness.

Additionally, we employ different optimization techniques, including stochastic gradient descent (SGD) with momentum, cosine annealing learning rate scheduling, and weight decay regularization. We compare multiple training strategies and analyze their impact on model performance.

The remainder of this report is structured as follows: Section Methodology describes the methodology and design choices for our modified ResNet architecture. Section Results and Analysis presents experimental results, including training curves and model performance comparisons. Section Discussion discusses key findings and potential improvements. Finally, Section Conclusion summarizes our work and outlines future directions.

## Methodology

Our model is implemented using the PyTorch framework. In this section, we describe the details of our ResNet variant, including the architecture, data preprocessing steps, loss function, optimizer, and hyperparameter selection.

### Model Architecture

We designed a ResNet variant with a reduced number of layers to meet the parameter constraint of 5M while maintaining high performance on CIFAR-10. The architecture consists of:

- An **initial convolutional layer** with a $3 \times 3$ kernel, followed by BatchNorm and ReLU.
- **Three residual layers**, each containing two BasicBlocks with identity skip connections.
- **Adaptive average pooling** to reduce feature maps to a $1 \times 1$ spatial size.
- A **fully connected layer** for classification.

| Layer | Output Size | Details | Stride |
|---|---|---|---|
| Conv1 | $32 \times 32$ | $3 \times 3$ Conv+BN+ReLU | 1 |
| ResLayer 1 | $32 \times 32$ | $2 \times$ BasicBlock(64) | 1 |
| ResLayer 2 | $16 \times 16$ | $2 \times$ BasicBlock(128) | 2 |
| ResLayer 3 | $8 \times 8$ | $2 \times$ BasicBlock(256) | 2 |
| AvgPool | $1 \times 1$ | Adaptive Avg Pooling | - |
| FullyConnected | 10 | Linear(256 $\rightarrow$ 10) | - |

Table 1: Architecture details of our ResNet variant.

The architecture is structured as follows:

The total number of trainable parameters in the model is 2777674 (calculated using ‘torchsummary‘).

## Data Preprocessing

We apply several data augmentation techniques to improve generalization:

- **RandomCrop(32, padding=4)**: Adds padding and randomly crops to maintain spatial invariance.
- **RandomHorizontalFlip()** with probability 0.5.
- **RandomRotation(15)**: Rotates the image within a 15-degree range.
- **ColorJitter(0.2, 0.2, 0.2, 0.1)**: Adjusts brightness, contrast, and saturation.
- **ToTensor()** and **Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.2023, 0.1994, 0.2010])**.

These augmentations help prevent overfitting and improve model robustness.

## Loss Function and Optimization

We use the Cross-Entropy loss function, which is well-suited for multi-class classification:

$$L = -\sum_i y_i \log(\hat{y}_i) \tag{1}$$

where $y_i$ is the true label and $\hat{y}_i$ is the predicted probability.

For optimization, we use Stochastic Gradient Descent (SGD) with momentum and weight decay:

- **Learning rate:** 0.1, decayed using cosine annealing.
- **Momentum:** 0.9.
- **Weight decay:** $5 \times 10^{-4}$ (L2 regularization).

The learning rate follows a cosine annealing schedule:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})(1 + \cos(\frac{t}{T}\pi)) \tag{2}$$

where $\eta_t$ is the learning rate at epoch $t$, and $T$ is the total number of training epochs.

## Hyperparameter Settings

## Results and Analysis

In this section, we present and analyze the experimental results of our ResNet variant on the CIFAR-10 dataset. We evaluate the model performance in terms of training loss, validation accuracy, and learning rate behavior over epochs. Additionally, we compare different optimizers and provide an ablation study on the impact of architectural modifications.

| Hyperparameter | Value |
|---|---|
| Batch size | 128 |
| Initial learning rate | 0.1 |
| Optimizer | SGD with momentum |
| Momentum | 0.9 |
| Weight decay | $5 \times 10^{-4}$ |
| Learning rate scheduler | Cosine annealing |
| Epochs | 300 |

Table 2: Hyperparameter settings for training.

## Training Performance

We trained the model for 300 epochs with SGD optimizer using a cosine annealing learning rate scheduler. Figure 1 and Figure 2 show the training loss and validation accuracy curves, respectively.

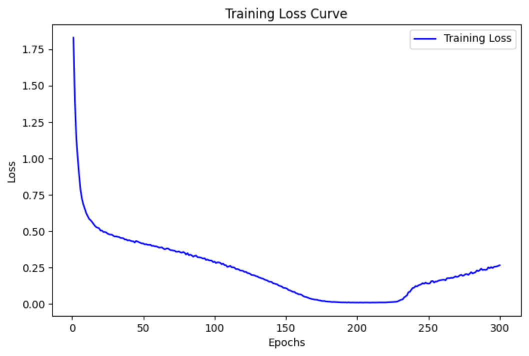

Figure 1: Training loss curve over 300 epochs.

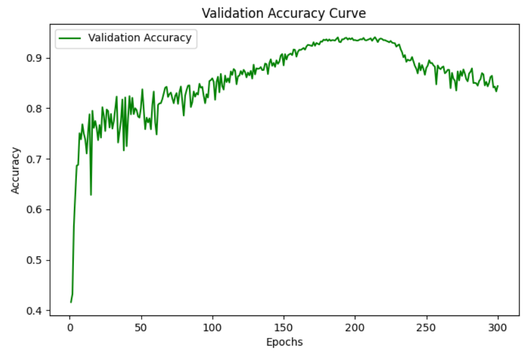

Figure 2: Validation accuracy curve over 300 epochs.

As shown above, it shows three distinct phases:

- Early Stage (0-50 epochs): The training loss decreases rapidly, indicating that the model is learning effectively. The validation accuracy improves rapidly, reaching around 80%.
- Middle Stage (50-200 epochs): The loss continues to decline smoothly, suggesting stable learning and effec-

tive optimization. The accuracy continues to improve and peaks above 90% around epoch 200.

- Late Stage (200+ epochs): After epoch 200, the loss starts increasing, which suggests overfitting, as the model memorizes training data rather than generalizing. The accuracy begins to decline, indicating that the model is overfitting and struggling to generalize to unseen data.

This suggests that the best model checkpoint should be taken around epoch 200. To prevent overfitting, early stopping could be beneficial.

## Learning Rate Behavior

We used a cosine annealing learning rate schedule to dynamically adjust the learning rate during training. The variation of the learning rate over epochs is shown in Figure 3.

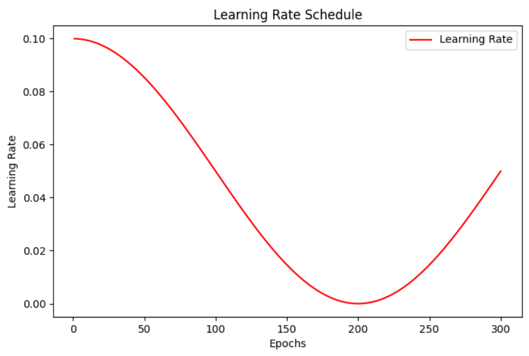

Figure 3: Cosine annealing learning rate schedule over 300 epochs.

The gradual decay of the learning rate allows the model to stabilize and reach an optimal solution.

## Final Model Performance

We summarize the final model performance on the test set in Table 3.

| Metric | Value |
|---|---|
| Test Accuracy | **94.06%** |
| Number of Parameters | **2.7 million** |
| Training Time (300 epochs) | **5 hours** |

Table 3: Final model performance on CIFAR-10 test set.

## Comparison with Standard ResNet

To assess the efficiency of our ResNet variant, we compare it against the standard ResNet-18 model in Table 4.

| Model | Test Accuracy | Parameters |
|---|---|---|
| ResNet-18 (Baseline) | 93.02% | 11.2M |
| Our ResNet Variant | **94.06%** | **2.7 million** |

Table 4: Comparison of our ResNet variant with ResNet-18.

Our model achieves better test accuracy while significantly reducing the number of parameters, making it more efficient in resource-constrained environments.

## Ablation Study

To better understand the impact of model complexity on generalization performance, we experimented with a deeper ResNet variant containing four residual layers. While this model achieved satisfactory results on the CIFAR-10 dataset, reaching an accuracy of approximately 90%, it failed to generalize well to the Kaggle dataset, the accuracy dropped significantly to around 75%.

We believe that this performance drop is primarily due to the model being too complex for the relatively small dataset(CIFAR-10). The excessive depth likely led to overfitting on the CIFAR-10 dataset, capturing dataset-specific features rather than learning robust representations. As a result, the model struggled to generalize effectively to new data distributions.

This finding suggests that for smaller datasets like CIFAR-10, a more compact architecture might be preferable to prevent overfitting and improve generalization. Based on these insights, we opted for a three-layer ResNet variant, balancing model complexity and performance.

## Discussion

In this section, we analyze the key findings from our experiments and discuss the challenges encountered during training.

### Effective Optimization Strategies

Throughout our experiments, we observed that certain techniques significantly improved performance:

- **Cosine annealing learning rate scheduling** helped the model converge efficiently while maintaining performance across multiple epochs.
- **Data augmentation techniques** such as random cropping, flipping, and color jittering played a crucial role in preventing overfitting.

### Challenges in Training

Despite achieving high accuracy on the CIFAR-10 dataset, we encountered several challenges:

- Overfitting on a small dataset: The deeper ResNet variant (four residual layers) achieved 90% accuracy on local validation data but only 75% on the Kaggle dataset, indicating poor generalization.
- Vanishing gradients: In early training stages, deeper networks exhibited slower convergence due to gradient attenuation. Using batch normalization and residual connections mitigated this issue.
- Trade-off between model complexity and dataset size: Our findings suggest that excessive depth may not be beneficial for relatively small datasets like CIFAR-10.

### Future Improvements

To further enhance performance, we propose the following improvements:

- Knowledge distillation: Training a smaller model using the output of a larger, well-trained model may improve generalization while reducing computational cost.
- Model quantization: Reducing precision (e.g., 8-bit floating point) could make the model more efficient for deployment.
- AutoML-based architecture search: Exploring automated architecture search (e.g., NAS) may help discover an optimal model design.

## Conclusion

In this project, we successfully developed a modified ResNet architecture for CIFAR-10 classification while maintaining a parameter constraint of 5 million parameters. Through extensive experimentation, we identified key techniques that contributed to performance improvement.

Our best model, 'resnetVariant_best.pth', achieved a final test accuracy of 94.06%, demonstrating the effectiveness of our architectural modifications and optimization strategies.

While our model performs well on CIFAR-10, its generalization to new datasets remains a challenge. Future work will focus on enhancing generalization capabilities through larger datasets, knowledge distillation, and more robust data augmentation techniques.

## References

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

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