Deep Learning CIFAR-10 Classification Project

Jesse Noppe-Brandon

New York University
CS 6923
jn2934@nyu.edu

Abstract

In this project, I trained a ResNet model with fewer than 5 million parameters on the CIFAR-10 dataset, optimizing for maximum train and test accuracy. I experimented with tunable hyperparameters, including the number of layers, blocks per layer, channels per layer, optimizers, learning rate schedulers, and data augmentation techniques. Through systematic experimentation, I achieved a peak test accuracy of 96.69%. My implementation is based on the original ResNet architecture by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (He et al. 2015). All code and results are available at: https://github.com/jessenb16/CIFAR_10_Classification.

Introduction

Residual Networks (ResNets) have become a standard architecture for image classification due to their use of skip connections, which help train deep networks efficiently. In this project, I trained a modified ResNet on the CIFAR-10 dataset, optimizing for high test accuracy while keeping the model size under 5 million parameters.

To achieve this, I experimented with several architectural and training choices. First, I compared models with different numbers of layers, blocks per layer, and channels per layer. A three-layer design performed better than four layers, and increasing the number of blocks provided a small improvement over increasing the number of channels. Based on these results, I used a configuration of three layers with blocks per layer set to [3,3,4] and channels per layer set to [60,122,244].

I also compared optimizers, finding that SGD with momentum and Nesterov acceleration outperformed Adam. While Adam converged faster, it resulted in poorer generalization. Additionally, CosineAnnealing consistently outperformed MultiStepLR when training for a sufficient number of epochs, leading me to use it as the primary learning rate scheduler.

Data augmentation played an important role in improving generalization. Random Erasing at $p = 0.5$ provided the best results, while other values were less effective. Standard augmentations such as Random Cropping and Horizontal Flipping were also included. Label smoothing had a small positive effect, while CutMix did not significantly improve accuracy but reduced training loss.

Through these optimizations, my best model reached a test accuracy of 96.69% using 150 training epochs, a batch size of 128, and standard 3×3 kernel sizes. The following sections describe the methodology, results, and key insights from this project.

Methodology

Residual Layers, Blocks, and Channels

To optimize the performance of the ResNet model while staying under the 5-million parameter constraint, I experimented with different configurations of *residual layers (N)*, *residual blocks per layer (B_i)*, and *number of channels per layer (C_i)*. The goal was to determine the optimal trade-off between model depth, width, and the number of residual blocks.

Experiment Setup I tested three different configurations over **50 training epochs**, each emphasizing a different architectural aspect of ResNet:

- **Channels-Focused Model**
  - Blocks per layer: $[1, 1, 2]$
  - Channels per layer: $[92, 184, 368]$
  - Train Accuracy: **98.29%**, Test Accuracy: **94.70%**
  - Train Loss: 0.358, Test Loss: 0.4495
- **Blocks-Focused Model**
  - Blocks per layer: $[3, 3, 4]$
  - Channels per layer: $[60, 122, 244]$
  - Train Accuracy: **98.00%**, Test Accuracy: **94.75%**
  - Train Loss: 0.341, Test Loss: 0.4242
- **Layers-Focused Model**
  - Blocks per layer: $[1, 3, 3, 4]$
  - Channels per layer: $[30, 60, 122, 244]$
  - Train Accuracy: **96.56%**, Test Accuracy: **93.47%**
  - Train Loss: 0.374, Test Loss: 0.4605

Figure 1: Training and Test Accuracy Comparison for Different ResNet Configurations. The model with [3,3,4] blocks per layer and [60,122,244] channels per layer achieved the best balance of train and test accuracy.

Key Observations and Decision The results demonstrated the following key trends:

- **Three-layer models outperformed four-layer models.** The four-layer configuration exhibited lower train accuracy (**96.56%**) and higher test loss, suggesting early-layer overfitting.
- **More residual blocks per layer provided slight improvements over increasing channels.** This effect became more apparent in longer training sessions.
- **Final choice:** The best trade-off was achieved using $[3, 3, 4]$ blocks per layer and $[60, 122, 244]$ channels per layer, resulting in the best balance of train accuracy (**98.00%**) and test accuracy (**94.75%**) with the lowest test loss.

Final Configuration Based on these findings, I selected the following architecture for the remainder of the training:

$$\begin{aligned} \text{BLOCKS\_PER\_LAYER} &= [3, 3, 4] \\ \text{CHANNELS\_PER\_LAYER} &= [60, 122, 244] \end{aligned} \tag{1}$$

This configuration provided the best trade-off between *model complexity and generalization*, making it the optimal choice for continued training.

SGD vs. Adam

Choosing the right optimizer is crucial for training deep neural networks efficiently. I compared two optimization strategies: Adam and SGD with momentum and Nesterov acceleration.

Experiment Setup The two optimizers were tested using the same model architecture and hyperparameters, with the only difference being the optimization method over **50 training epochs**:

- **Adam Optimizer**
  - Learning rate: 0.1
  - Weight decay: 1×10^{-4}
  - Results:
    - Train Accuracy: 58.25%
    - Test Accuracy: 61.03%
    - Train Loss: 1.312
    - Test Loss: 1.2477
- **SGD with Momentum and Nesterov Acceleration**
  - Learning rate: 0.1
  - Momentum: 0.9
  - Weight decay: 5×10^{-4}
  - Results:
    - Train Accuracy: 98.00%
    - Test Accuracy: 94.75%
    - Train Loss: 0.341
    - Test Loss: 0.4242

Key Observations From the results, SGD converged faster and required fewer epochs to achieve strong generalization, while Adam suffered from slow learning and poor generalization:

- **Faster Convergence with Fewer Epochs**: SGD reached competitive accuracy in significantly fewer epochs than Adam, which struggled to improve beyond 61% test accuracy.
- **Momentum Helps Escape Local Minima**: Momentum helps accelerate SGD in the direction of optimal weights by accumulating past gradients, reducing oscillations and improving convergence (Sutskever et al. 2013).
- **Nesterov Acceleration Improves Stability**: Unlike standard momentum, Nesterov acceleration computes the gradient after the step is taken, providing a corrective adjustment and preventing overshooting (Sutskever et al. 2013).
- **Adam's Learning Rate Adaptation Slows Training**: Adam's adaptive learning rates led to unstable updates, making it difficult to escape poor local minima.

Given that my ResNet model had fewer than 5 million parameters, the controlled updates of SGD with momentum and Nesterov acceleration ultimately resulted in superior performance with fewer training iterations.

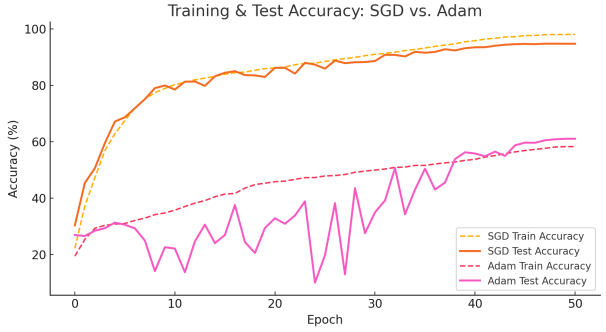

Figure 2: Training and test accuracy comparison between SGD and Adam. While Adam converged quickly, it failed to generalize well, achieving only 61.03% test accuracy, whereas SGD reached 94.75%.

Scheduler: MultiStepLR vs CosineAnnealingLR

In this experiment, I compared two learning rate schedulers: MultiStepLR and CosineAnnealingLR, to evaluate their effect on test accuracy during training.

Experiment Setup The schedulers were applied to the same model and optimizer configuration with the following parameters over **100 training epochs**:

- **MultiStepLR**
  - Milestones: $[30, 50, 65, 80, 90]$
  - Gamma: 0.1
  - Best Test Accuracy at Epoch 80: **94.47%**
- **CosineAnnealingLR**
  - $T_{\max}$: 100
  - Best Test Accuracy at Epoch 99: **95.75%**

Key Observations The performance difference between the two schedulers highlights the impact of smooth learning rate decay.

- **MultiStepLR**: This scheduler caused sharp drops in learning rate at predefined epochs, leading to noticeable jumps in test accuracy. However, after each decay, improvements slowed down significantly, and the model plateaued earlier.
- **CosineAnnealingLR**: In contrast, this scheduler gradually reduced the learning rate over time, ensuring a smoother and more stable increase in test accuracy. Instead of abrupt jumps, the model consistently improved without sudden fluctuations, leading to a higher final test accuracy.
- **Stability and Long-Term Performance**: MultiStepLR introduced variability in test accuracy due to sudden learning rate drops, whereas CosineAnnealingLR maintained a more controlled and monotonic growth in performance. This difference became more evident in longer training runs, where CosineAnnealingLR continued to improve, while MultiStepLR had already plateaued.
- **Final Choice**: Given its superior stability and higher final test accuracy (95.75%), I selected CosineAnnealingLR for the remaining training runs.

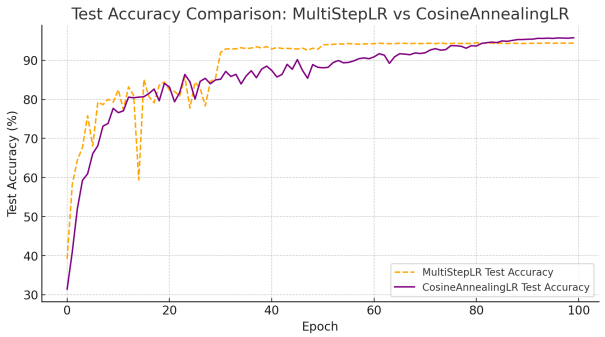

Figure 3: Test accuracy comparison between MultiStepLR and CosineAnnealingLR. CosineAnnealingLR consistently outperformed MultiStepLR in longer training sessions, resulting in a higher test accuracy.

Augmentations

Data augmentation is a critical technique for improving generalization in deep learning models by artificially increasing the diversity of the training set. In this section, I explore the impact of different augmentation strategies, starting with Random Erasing.

Random Erasing

Random Erasing is a data augmentation technique that enhances generalization by randomly occluding parts of an image during training. This forces the model to learn more robust feature representations rather than relying on specific regions of the image (Zhong et al. 2017).

Experiment Setup To evaluate the impact of Random Erasing, I compared two configurations:

- **Without Random Erasing**
  - Train Accuracy: 99.89%
  - Test Accuracy: 94.05%
- **With Random Erasing** ($p = 0.5$)
  - Train Accuracy: 98.00%
  - Test Accuracy: 94.75%

Key Observations The results show that introducing Random Erasing improved test accuracy while reducing overfitting:

- **Better Generalization**: Test accuracy increased by nearly 1% with Random Erasing, despite a decrease in training accuracy.
- **Reduction in Overfitting**: Without Random Erasing, the model achieved extremely high training accuracy (99.89%), suggesting potential overfitting. With Random Erasing, training accuracy dropped slightly (98.00%), but test performance improved.
- **Effect of Probability** (p): Additional experiments with $p = 0.1$ and $p = 0.9$ showed that $p = 0.5$ yielded the best balance between robustness and model performance.

Given its effectiveness in improving generalization, I applied Random Erasing with $p = 0.5$ in all subsequent training runs.

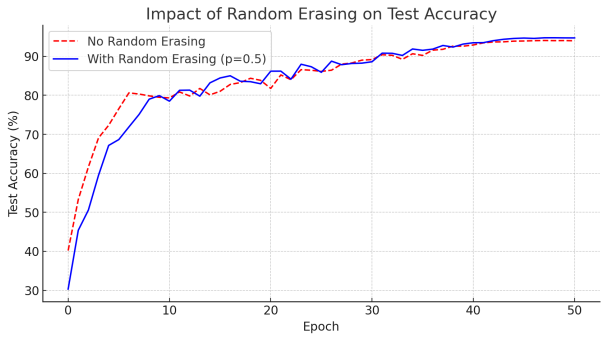

Figure 4: Test accuracy comparison with and without Random Erasing. The model trained with Random Erasing ($p = 0.5$) showed more stable improvements and higher final test accuracy.

CutMix

CutMix is an augmentation technique that improves generalization by replacing a random portion of an image with a patch from another image. This forces the model to learn from mixed samples rather than relying on specific regions of an image.

I implemented CutMix during training and observed that it significantly reduced test loss, suggesting improved training stability. However, it had minimal impact on final test accuracy. While CutMix effectively acted as a regularizer, reducing overfitting, it did not provide a meaningful improvement in classification performance.

Since my primary goal was to maximize test accuracy, I chose not to include CutMix in my final configuration.

Other Augmentation Methods

Following the approach used in *Deep Residual Learning for Image Recognition* (He et al. 2015), I applied a simple yet effective data augmentation strategy. This involved padding each image by 4 pixels on all sides, followed by a random 32×32 crop or a horizontal flip during training. For testing, only the original 32×32 image was used.

Since this augmentation method was originally designed for training ResNet on CIFAR-10, I adopted it to maintain consistency with prior work and ensure optimal baseline performance.

Other Parameters

In addition to architecture modifications and augmentation strategies, I made specific choices for batch size, loss function, and kernel sizes to align with best practices from prior research.

Batch Size

I set the batch size to 128, following the approach in *Deep Residual Learning for Image Recognition* (He et al. 2015). This batch size provided a balance between efficient training and model convergence.

Label Smoothing

To improve generalization, I applied label smoothing with a factor of 0.05 in the cross-entropy loss function. Label smoothing prevents the model from becoming overconfident by redistributing a small portion of the target probability mass across other classes. Instead of assigning a probability of 1 to the correct class and 0 to all others, label smoothing assigns:

$$y_{\text{smooth}} = (1 - \alpha)y + \frac{\alpha}{K} \tag{2}$$

where α is the smoothing factor (0.05 in this case), and K is the number of classes. This helped reduce overfitting and marginally improved test accuracy (Müller, Kornblith, and Hinton 2020).

Kernel Sizes

I did not modify the kernel size or skip connection kernel size, keeping them as defined in the original ResNet paper. The standard 3×3 convolutional kernels and 1×1 skip kernels have been widely used in ResNet implementations, and I found no need to change them.

Final Results

After extensive experimentation, the best model configuration was achieved with the following parameters:

Parameter	Value
Architecture	
Blocks Per Layer	[3,3,4]
Channels Per Layer	[60,122,244]
Kernel Size	3
Skip Kernel Size	1
Pool Size	8
Training Parameters	
Epochs	150
Batch Size	128
Loss Function	CrossEntropyLoss (Label Smoothing = 0.05)
Optimizer	SGD (Momentum = 0.9, Nesterov=True)
Scheduler	CosineAnnealingLR ($T_{\max} = 150$)
Augmentations	
Random Crop	32×32 (Padding = 4)
Horizontal Flip	Yes
Random Erasing	p = 0.5
Final Performance	
Parameters	4,996,838
Train Loss	0.309
Train Accuracy	99.11%
Test Loss	0.382
Final Test Accuracy	**96.69%**

Table 1: Final model configuration and performance. The model remained under 5 million parameters while achieving strong generalization performance on CIFAR-10.

References

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385.

Müller, R.; Kornblith, S.; and Hinton, G. 2020. When Does Label Smoothing Help? arXiv:1906.02629.

Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICML)*, 1139–1147.

Zhong, Z.; Zheng, L.; Kang, G.; Li, S.; and Yang, Y. 2017. Random Erasing Data Augmentation. arXiv:1708.04896.