CustomResNet: A Mini ResNet Architecture for CIFAR-10 Classification

Nick Ferrara

Deep Learning
Professor Hegde
github.com/NxFerrara/cifar-10-classification

Abstract

In this project, we developed a custom ResNet architecture for CIFAR-10 image classification with less than 5 million parameters. Our approach focuses on channel allocation in residual blocks, with a configuration of [48, 96, 128, 364] channels across four layers. Using data augmentation techniques and the OneCycleLR learning rate scheduler, our model achieved 94.18% accuracy on the test set, demonstrating that even a miniaturized model can perform well.

Introduction

Image classification is fundamental in computer vision, with CIFAR-10 serving as a standard benchmark. This project addresses the challenge of developing a classifier within a strict parameter budget of 5 million parameters, using residual networks (ResNets) as the foundation.

ResNets address the vanishing gradient problem in deep neural networks through skip connections, allowing effective training of deeper layers.

The CIFAR-10 dataset [2] consists of 60,000 32×32 color images across 10 classes.

In this report, we detail our approach to designing an efficient ResNet architecture, our training methodology, and the performance achieved. Our final model obtains 94.18% accuracy while using 4,997,722 million parameters.

Methodology

Model Architecture

Our architecture is based on the ResNet-18 framework, with modifications to attempt to maintain performance while reducing parameter count. The architecture includes:

- An initial 3×3 convolutional layer with 48 output channels
- Four residual layers with channel configurations [48, 96, 128, 364]
- Each residual layer contains two residual blocks
- Standard skip connections with projection shortcuts where dimensions change
- A fixed 4×4 average pooling layer followed by a fully-connected layer

Figure 1 illustrates the overall architecture of our model.

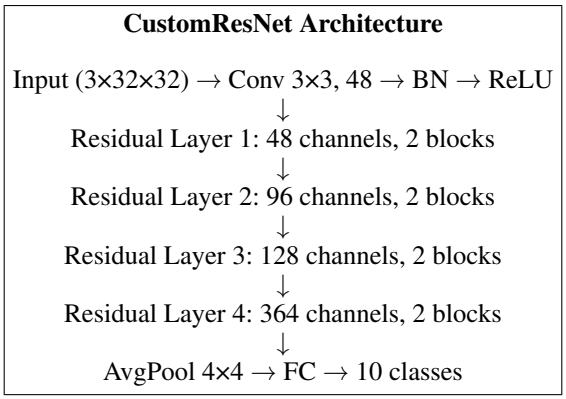

Figure 1: Overview of our architecture showing the layer structure and channel configurations.

Each residual block follows the standard formula:

$$\text{Output} = \text{ReLU}(S(x) + F(x))$$

where S(x) is the skip connection path and F(x) is the residual function implementing conv → BN → ReLU → conv → BN.

During our design process, we learned that using narrower channels in early layers and progressively increasing them was more effective than distributing parameters more evenly, which makes sense since the narrower layers would capture macro features as the larger layers deal with the idiosyncrasies.

Parameter Efficiency

Our model contains a total of 4,997,722 million parameters. We verified this parameter count using the torchsummary.summary function as recommended in the project guidelines. Table 1 shows the parameter distribution across different components of the network.

One key design lesson we learned was the importance of regularization since overfitting was a large issue. Adding a dropout at the final layer of 10% helped tremendously. Additionally, our initial attempts allocated too many parameters

to early layers, resulting in what seems like wasted capacity. We found that a larger factor than 8 between the first and final layer seems to perform better.

Training Approach

We employed several techniques to effectively train our model:

- **Data Augmentation:** Random crops with padding of 4 pixels and random horizontal flips to increase dataset variety
- **Optimization:** SGD optimizer with an initial learning rate of 0.1, momentum of 0.9, and weight decay of 5e-4
- **Learning Rate Schedule:** OneCycleLR policy with a maximum learning rate of 0.1 and 10% warmup period
- **Regularization:** Weight decay, batch normalization, and dropout (p=0.1) to prevent overfitting
- **Early Stopping:** Training halts after 10 epochs without validation accuracy improvement

We split the original training data into a training set (90%) and a validation set (10%) to monitor model performance during training. The model was trained for up to 100 epochs with a batch size of 128. [2]

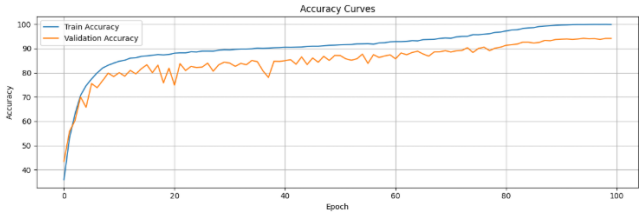

Figure 2: Training and validation accuracy/loss over epochs. The model converges at epoch 94 with validation accuracy stabilizing at 94.18%.

Results

Model Performance

Our CustomResNet achieved 94.18% accuracy on the CIFAR-10 test set. Figure 3 shows the confusion matrix, highlighting the model's performance across different classes.

Analysis of the confusion matrix reveals that the model struggles most with distinguishing between visually similar categories like cats and dogs, which makes sense.

Table 2 summarizes the per-class precision, recall, and F1-scores:

Additional Findings

Notably, during our model development, we found that adding a small dropout (p=0.1) just before the fully connected layer increased accuracy by a few percentage points. This additional regularization helped prevent overfitting, especially given our parameter constraints.

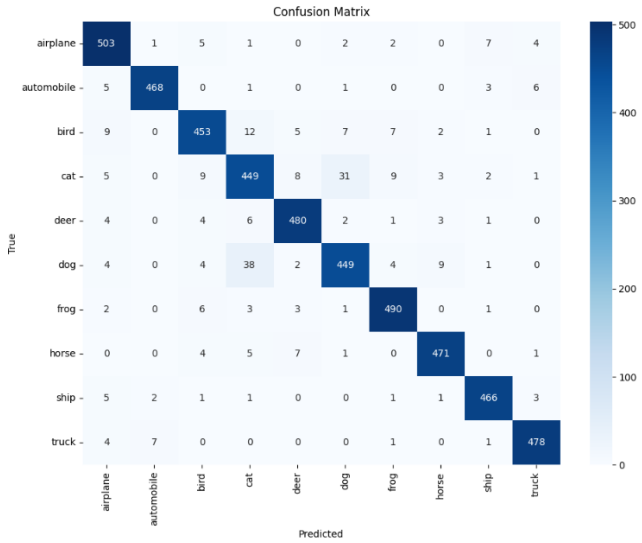

Figure 3: Confusion matrix on the CIFAR-10 test set. The model performs best on trucks/automobiles (98%), while showing lower performance on cats (88%) and dogs (90%).

Class	Precision	Recall	F1-Score
Airplane	0.92	0.96	0.94
Automobile	0.98	0.97	0.97
Bird	0.92	0.91	0.91
Cat	0.88	0.86	0.87
Deer	0.95	0.95	0.95
Dog	0.90	0.89	0.90
Frog	0.95	0.97	0.96
Horse	0.95	0.96	0.96
Ship	0.97	0.97	0.97
Truck	0.98	0.96	0.97
Average	**0.94**	**0.94**	**0.94**

Table 1: Per-class precision, recall, and F1-scores on the CIFAR-10 validation dataset. The model achieves an overall accuracy of 94%.

Conclusion

In this project, we successfully designed and trained a custom ResNet architecture for CIFAR-10 classification with fewer than 5 million parameters. Our model achieves 94.18% accuracy through our design choices in architecture and training methodology.

The key insights from our work include:

- Strategic channel allocation is critical for parameter efficiency—our progressive channel strategy with [48, 96, 128, 364] configuration proved effective for balancing model size and performance
- Proper regularization through batch normalization and dropout provides significant benefits for parameter-constrained models
- Modern learning rate scheduling techniques like OneCycleLR help achieve better convergence and final accuracy

Our results demonstrate that it's possible to achieve strong performance on CIFAR-10 with a relatively compact model.

Future work could explore alternative block designs such as increasing the size of residual layers, residual block kernel sizes, or additional regularization to further improve performance within the parameter constraints.

References

[1] Zhang, A.; Lipton, Z. C.; Li, M.; and Smola, A. J. 2023. Dive into Deep Learning, 1st Edition. Cambridge University Press.

[2] Beji, V. 2025. Deep Learning Spring 2025: CIFAR 10 classification. https://kaggle.com/competitions/deep-learning-spring-2025-project-1. Kaggle.