Deep Learning Mini-Project – Spring-25 – NYU – Image Classification: Enhancing CIFAR-10 Performance With Modified ResNet

Mini-Project GitHub Repository

Bhagya Rekha Deenadayal*, Yaswanth Kumar Damarla*, Bhanu Dileep Reddy Maryada*
New York University, Tandon School of Engineering, Department of Electrical and Computer Engineering (ECE)
6 MetroTech Center, Brooklyn, NY 11201

Abstract

In this report, we propose training a modified Residual Network (ResNet) architecture to achieve the highest possible test accuracy on the CIFAR-10 [1] image classification dataset, under the constraint that the total number of parameters does not exceed 5 million.

Introduction

Deep convolutional neural networks (CNNs) have significantly advanced the field of image classification. Among these architectures, ResNet-18 is a CNN model characterized by its 18-layer structure. In this project, we explore modifications to the ResNet-18 architecture to achieve an error rate below 10% on the CIFAR-10 image classification dataset. Specifically, we propose a residual learning approach that substantially reduces the number of trainable parameters compared to previous implementations. The trained model's performance was assessed using a separate test set containing images not previously encountered from the CIFAR-10 dataset. Evaluation was conducted using test accuracy and test loss metrics. Ultimately, our model achieved a test accuracy of approximately 84.161%, utilizing only 1.18 million trainable parameters.

Related Work

Training deep neural networks has long been a challenge due to issues such as vanishing gradients and optimization difficulties. In [2], researchers introduced a residual learning framework to simplify the training of significantly deeper networks compared to earlier models. Rather than directly learning the target function, their approach focuses on learning residual functions relative to the input of each layer. This architectural adjustment enhances optimization in deep networks. Their findings further demonstrate that Residual Networks (ResNets) not only optimize more effectively but also achieve better accuracy as depth increases. On the ImageNet dataset, a ResNet model with 152 layers attained a top-5 error rate of 3.57%, outperforming models like VGG.

The study *Efficient ResNets: Residual Network Design* [5] explores residual network architectures with fewer than 5 million parameters. The authors demonstrate that their ResNet variant, containing under 5 million parameters, achieves a test accuracy of 96.04% on CIFAR-10. This significantly surpasses the performance of ResNet-18, which has over 11 million trainable parameters, when combined with optimized training strategies and carefully selected ResNet hyperparameters.

Architecture

The CIFAR-10 dataset consists of 32x32 RGB images, which are relatively small compared to larger datasets such as ImageNet (224x224). To efficiently process such images, our model adopts a modified ResNet architecture inspired by CifarResNet [3]. The primary objective of our architecture is to balance performance and parameter efficiency, rather than achieving state-of-the-art results. As a result, we opted for a simple yet robust architecture that leverages residual connections and dropout regularization.

The model begins with an initial convolutional layer (3x3 kernel, 32 filters), followed by batch normalization and a ReLU activation function. This combination helps stabilize training and accelerate convergence. To enhance the robustness of the model, dropout layers are strategically incorporated throughout the architecture.

The architecture consists of a series of residual blocks, each containing two convolutional layers with batch normalization and ReLU activations. Dropout is applied after each convolution to mitigate overfitting. These residual blocks enable efficient training of deeper networks by leveraging shortcut connections.

To handle increasing feature map sizes, the architecture incorporates downsampling using strided convolutions. The feature map dimensions transition from 32x32 to 16x16 and finally to 8x8 as the number of filters increases from 32 to 64 and subsequently to 128. The global average pooling layer reduces the spatial dimensions to a 1x1 feature map, which is then flattened and passed through a fully connected layer to produce the final output of 10 classes.

The entire model comprises 108 layers, including convolutional, batch normalization, dropout, and residual connections. The total number of parameters is approximately 1.18 million. The model is trained using stochastic gradient de-

scent (SGD) with momentum, a learning rate of 0.09, and weight decay for regularization. To dynamically adjust the learning rate, we employ a multi-step learning rate scheduler.

Layer (type)	Output Shape	Param #
Input Layer		
Conv2d-1	[-1, 32, 32, 32]	864
BatchNorm2d-2	[-1, 32, 32, 32]	64
ReLU-3	[-1, 32, 32, 32]	0
Intermediate Layers		
Conv2d-4	[-1, 32, 32, 32]	9,216
BatchNorm2d-5	[-1, 32, 32, 32]	64
ReLU-6	[-1, 32, 32, 32]	0
Dropout-7	[-1, 32, 32, 32]	0
Conv2d-8	[-1, 32, 32, 32]	9,216
BatchNorm2d-9	[-1, 32, 32, 32]	64
Dropout-10	[-1, 32, 32, 32]	0
ReLU-11	[-1, 32, 32, 32]	0
BasicBlock-12	[-1, 32, 32, 32]	0
Conv2d-13	[-1, 64, 16, 16]	18,432
BatchNorm2d-14	[-1, 64, 16, 16]	128
ReLU-15	[-1, 64, 16, 16]	0
Dropout-16	[-1, 64, 16, 16]	0
Conv2d-17	[-1, 64, 16, 16]	36,864
BatchNorm2d-18	[-1, 64, 16, 16]	128
Dropout-19	[-1, 64, 16, 16]	0
ReLU-20	[-1, 64, 16, 16]	0
BasicBlock-21	[-1, 64, 16, 16]	0
Conv2d-22	[-1, 128, 8, 8]	73,728
BatchNorm2d-23	[-1, 128, 8, 8]	256
ReLU-24	[-1, 128, 8, 8]	0
Dropout-25	[-1, 128, 8, 8]	0
Conv2d-26	[-1, 128, 8, 8]	147,456
BatchNorm2d-27	[-1, 128, 8, 8]	256
Dropout-28	[-1, 128, 8, 8]	0
ReLU-29	[-1, 128, 8, 8]	0
BasicBlock-30	[-1, 128, 8, 8]	0
⋮	⋮	⋮
BasicBlock-106	[-1, 128, 8, 8]	0
AdaptiveAvgPool2d-107	[-1, 128, 1, 1]	0
Output Layer		
Linear-108 (Fully Connected)	[-1, 10]	1,290
Total Parameters		1,177,130
Trainable Parameters		1,177,130
Non-Trainable Parameters		0

Figure 1: Model summary

Figure 1 demonstrates the model architecture and an effective combination of residual learning and regularization techniques to achieve competitive performance on CIFAR-10 while maintaining parameter efficiency.

Methodology

Data augmentation is a critical technique used to artificially expand the size and diversity of a training dataset by applying various transformations to the input images. This helps the model generalize better to unseen data by simulating real-world variations. Below is an overview of the methods implemented in the provided code and their goals:

Random Horizontal Flip: This transformation flips the image horizontally with a 50% probability. It reflects the possibility of objects appearing in either orientation in real-world scenarios, such as a car facing left or right. By introducing this variation, the model becomes invariant to horizontal orientation, improving its robustness to such changes.

Random Crop with Padding: This technique involves cropping a random region of the image and padding it back to its original size. In the code, the image is cropped to a size of 32×32 pixels with a padding of 4 pixels. This simulates zooming in on different parts of the image, encouraging the model to focus on smaller details and learn features from various regions of the image.

Normalization: The pixel values of the image are normalized using the mean and standard deviation values specific to the dataset:

$$mean = [0.4914, 0.4822, 0.4465],$$
$$std = [0.247, 0.243, 0.261].$$

Normalization ensures that the input data has a consistent scale, which accelerates training and prevents certain features from dominating the learning process. This step is essential for improving the model's convergence and overall performance.

These transformations are carefully chosen to simulate real-world variations in image data, such as changes in orientation, position, and pixel intensity. By applying these augmentations, the model is trained to focus on invariant features that define the object classes in the CIFAR-10 dataset. This leads to improved generalization and better performance on unseen data, reducing the risk of overfitting.

In this project, we employed specific methods for optimization, loss calculation, and learning rate scheduling. These choices are integral to achieving optimal performance and stability during training. Below, we discuss the rationale behind these design decisions:

Cross Entropy Loss: The `CrossEntropyLoss` function from PyTorch is utilized as our primary loss function. It is well-suited for multi-class classification tasks like CIFAR-10, where each image belongs to one of ten distinct classes. This loss function calculates the difference between the predicted probability distribution and the actual labels, penalizing incorrect predictions. Cross-entropy loss effectively guides the model to assign higher probabilities to the correct class, making it particularly effective for deep learning tasks with categorical outputs.

Optimizer - Stochastic Gradient Descent (SGD): We chose Stochastic Gradient Descent (SGD) with momentum as our optimizer, configured with:

- Learning rate: 0.09
- Momentum: 0.9
- Weight decay: 3×10^{-4}

The momentum term helps accelerate convergence by mitigating oscillations in the gradient descent path, while weight decay helps reduce overfitting by adding a regularization term. SGD is computationally efficient and highly effective when used with large datasets, making it a fitting choice for our CIFAR-10 classification task.

Learning Rate Scheduler - MultiStepLR: Our learning rate is managed using the `MultiStepLR` scheduler,

which reduces the learning rate by a factor of 0.1 at predefined epochs: $[80, 120, 150, 180]$. This approach helps the model converge more smoothly by gradually reducing the step size, allowing for better fine-tuning during the later stages of training.

The strategy of decaying the learning rate at specific milestones helps balance between exploration and exploitation, effectively avoiding sharp drops in accuracy and promoting convergence toward a robust solution.

Results

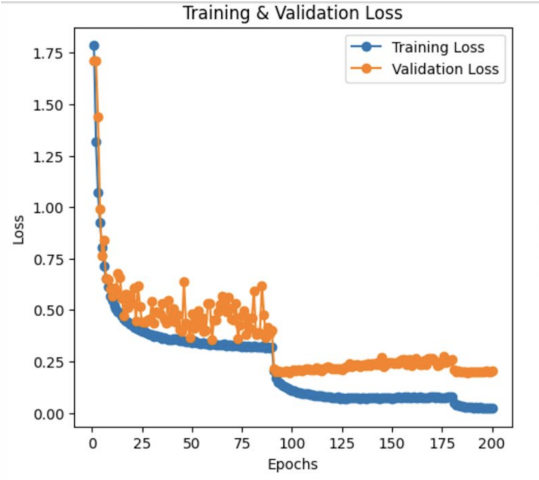

Figure 2: Training and Validation Loss

As depicted in Figure 2, the training and validation loss curves are plotted against each epoch. Our ResNet architecture achieved convergence with a final training loss of 0.216 and a validation loss of 0.2021.

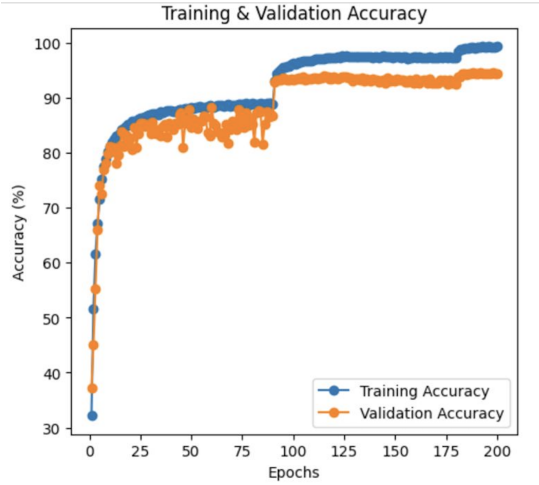

Figure 3: Training and Validation Accuracy

Figure 3 illustrates the progression of training and validation accuracy plotted against each epoch. Our ResNet implementation reached convergence with a final training accuracy of 99.41% and a validation accuracy of 94.47%.

The training process was limited to 200 epochs, as we noted that the model had reached convergence at this stage. To prevent overfitting to the training dataset, we deliberately terminated the training process at 200 epochs rather than continuing further.

References

[1] A. Krizhevsky, "Learning multiple layers of features from tiny images," Technical Report, University of Toronto, 2009.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CoRR*, abs/1512.03385, 2015.

[3] S. Das, "Implementation of ResNet Architecture for CIFAR-10 and CIFAR-100 Datasets," 2023.

[4] W. Liu *et al.*, "Improvement of CIFAR-10 Image Classification Based on Modified ResNet-34," in *Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery. ICNC-FSKD 2020. Lecture Notes on Data Engineering and Communications Technologies*, vol. 88, Springer, Cham, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-70665-4_68

[5] A. Thakur, H. Chauhan, and N. Gupta, "Efficient ResNets: Residual Network Design," *arXiv preprint arXiv:2306.12100*, 2023.