Deep Learning Project 1
ResNet Optimization for Image Classification

Kirit Govindaraja Pillai, Ruochong Wang, Saketh Raman Ramesh
kx2222@nyu.edu, rw3760@nyu.edu, sr7714@nyu.edu
New York University
GitHub Repository

Abstract

In this project, we explore modifications to the ResNet architecture to achieve high classification accuracy while maintaining a constraint of fewer than 5 million trainable parameters. We experiment with various optimization techniques, including adaptive learning rates, efficient optimizers, layer modifications, and dropout strategies. Additionally, we leverage automated hyperparameter tuning to systematically analyze the impact of these changes. Our findings provide insights into balancing model performance and computational efficiency, which is crucial for resource-constrained applications.

Introduction

Deep learning models often face the challenge of balancing classification accuracy with computational efficiency, particularly in resource-constrained environments. In this project, we aim to optimize the ResNet architecture by implementing modifications that enhance performance while keeping the number of trainable parameters below 5 million. To achieve this, we explore various techniques, including adaptive learning rates, efficient optimizers, layer modifications, and dropout strategies.

To systematically analyze these changes, we utilize Weights & Biases (WandB), a platform for experiment tracking and model management. Specifically, we employ WandB Sweeps, an automated hyperparameter optimization tool that allows us to efficiently explore different configurations through strategies like grid search, random search, and Bayesian optimization. By parallelizing experiments across multiple machines and providing real-time insights, Sweeps help identify the most effective hyperparameters for balancing accuracy and efficiency.

ResNet

ResNet (Residual Network) is a deep convolutional neural network architecture designed to address the degradation problem in very deep networks, where performance saturates and then degrades as layers increase. The core idea behind ResNet is the introduction of residual learning through skip connections or shortcut connections. These connections bypass one or more layers by directly adding the input of a layer to its output, forming residual blocks.

Mathematically, instead of learning a direct mapping H(x), ResNet models learn the residual function H(x)=F(x)-x, so the original mapping becomes H(x)=F(x)+x. This helps preserve gradient flow during backpropagation, mitigating the vanishing gradient problem and enabling training of much deeper networks.

A typical ResNet architecture consists of convolutional layers, batch normalization, and ReLU activations, arranged in residual blocks. Different ResNet variants, such as ResNet-18, ResNet-34, ResNet-50, and ResNet-101, represent the number of layers in the model. Deeper versions use bottleneck designs to reduce computation while maintaining performance.

ResNet's introduction revolutionized deep learning, especially in image classification tasks, enabling deeper networks to achieve remarkable accuracy. Its residual learning concept has since become a fundamental technique in modern neural network design.

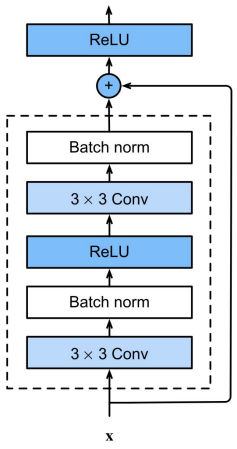

Figure 1: ResNet Identity Block

Proposed Model

To optimize training, the model was configured with a refined set of hyperparameters, including `num_epochs: 300`, `learning_rate: 0.0001`, `weight_decay: 0.001`, and `batch_size: 128`. The Adadelta optimizer was chosen due to its adaptive learning rate adjustments, making it well-suited for training deep neural networks. Additionally, cross-entropy loss (`nn.CrossEntropyLoss()`) was employed as the objective function, ensuring effective multi-class classification.

To enhance convergence, a learning rate scheduler (`ReduceLROnPlateau`) was implemented, dynamically adjusting the learning rate based on model performance. The scheduler monitored validation accuracy and reduced the learning rate by a factor of 0.5 if no improvement was observed for 10 consecutive epochs, helping to prevent overfitting and improve generalization. This combination of hyperparameter tuning and adaptive learning rate control contributed to stabilizing training and maximizing model performance.

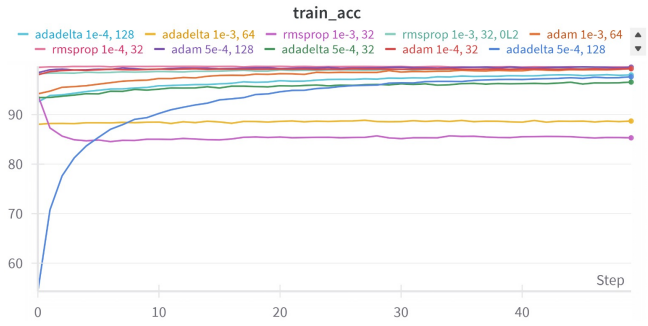

Figure 2: Training Accuracy

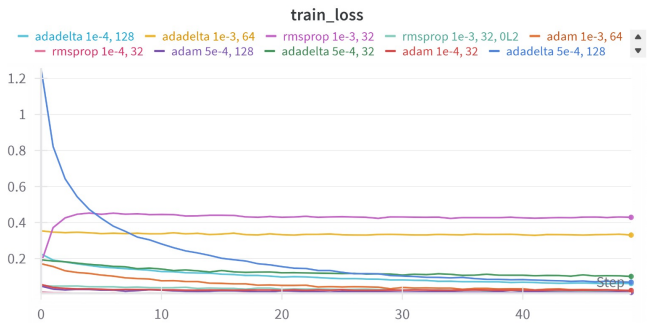

Figure 3: Training Loss

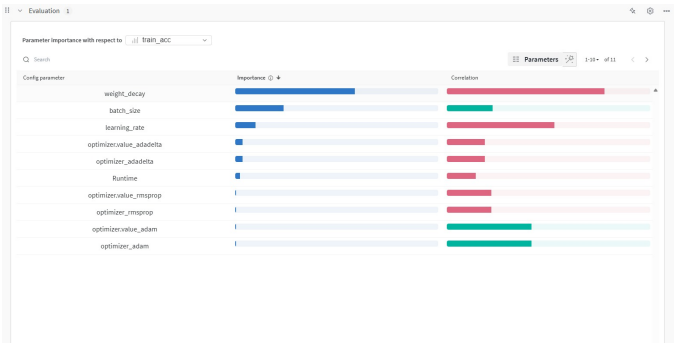

Figure 4: Hyperparameter Relevance

After running multiple trials, wandb identified the best-performing hyperparameters as follows: epochs: 50, optimizer: adadelta, batch_size: 128, weight_decay: 0.001, and learning_rate: 0.0001. These values yielded the highest validation accuracy, making them the optimal choice for training. Throughout the process, wandb.log() tracked training loss and accuracy at each epoch, enabling a clear comparison of different configurations. By integrating wandb, hyperparameter tuning was both efficient and insightful, leading to a well-optimized model.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 32, 32]	432
BatchNorm2d-2	[-1, 16, 32, 32]	32
Conv2d-3	[-1, 64, 32, 32]	9,216
BatchNorm2d-4	[-1, 64, 32, 32]	128
Dropout-5	[-1, 64, 32, 32]	0
Conv2d-6	[-1, 64, 32, 32]	36,864
BatchNorm2d-7	[-1, 64, 32, 32]	128
Dropout-8	[-1, 64, 32, 32]	0
Conv2d-9	[-1, 64, 32, 32]	4,096
BatchNorm2d-10	[-1, 64, 32, 32]	128
Block-11	[-1, 64, 32, 32]	0
Conv2d-12	[-1, 64, 32, 32]	36,864
BatchNorm2d-13	[-1, 64, 32, 32]	128
Dropout-14	[-1, 64, 32, 32]	0
Conv2d-15	[-1, 64, 32, 32]	36,864
BatchNorm2d-16	[-1, 64, 32, 32]	128
Block-17	[-1, 64, 32, 32]	0
Conv2d-18	[-1, 64, 32, 32]	36,864
BatchNorm2d-19	[-1, 64, 32, 32]	128
Dropout-20	[-1, 64, 32, 32]	0
Conv2d-21	[-1, 64, 32, 32]	36,864
BatchNorm2d-22	[-1, 64, 32, 32]	128
Block-23	[-1, 64, 32, 32]	0
Conv2d-24	[-1, 128, 16, 16]	73,728
BatchNorm2d-25	[-1, 128, 16, 16]	256
Dropout-26	[-1, 128, 16, 16]	0
Conv2d-27	[-1, 128, 16, 16]	147,456
BatchNorm2d-28	[-1, 128, 16, 16]	256
Dropout-29	[-1, 128, 16, 16]	0
Conv2d-30	[-1, 128, 16, 16]	16,384
BatchNorm2d-31	[-1, 128, 16, 16]	256
Block-32	[-1, 128, 16, 16]	0
Conv2d-57	[-1, 256, 8, 8]	294,912
BatchNorm2d-58	[-1, 256, 8, 8]	512
Dropout-59	[-1, 256, 8, 8]	0
Conv2d-60	[-1, 256, 8, 8]	589,824
BatchNorm2d-61	[-1, 256, 8, 8]	512
Dropout-62	[-1, 256, 8, 8]	0
Conv2d-63	[-1, 256, 8, 8]	65,536
BatchNorm2d-64	[-1, 256, 8, 8]	512
Block-65	[-1, 256, 8, 8]	0
AdaptiveAvgPool2d-78	[-1, 256, 1, 1]	0
Linear-79	[-1, 10]	2,570
Total Params		**4,934,746**
Trainable Params		**4,934,746**
Non-trainable Params		**0**

Evaluation

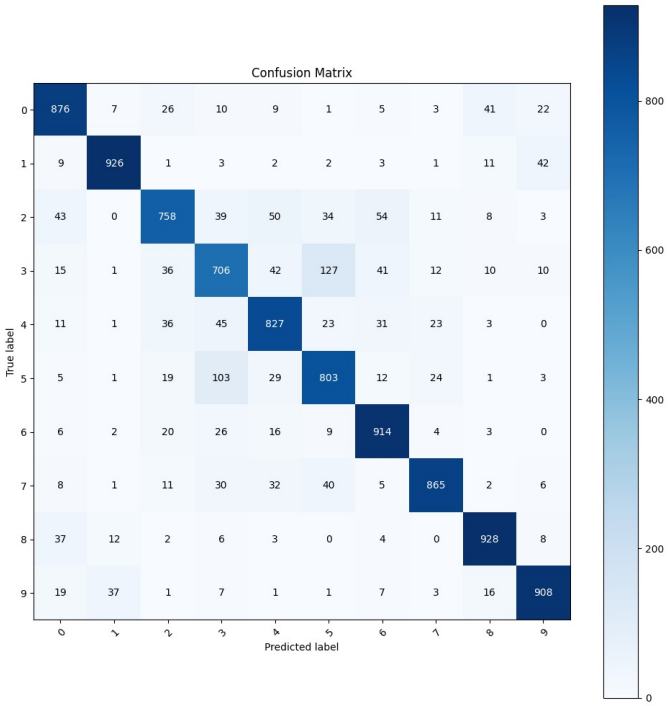

Figure 5: Confusion Matrix

Figure 6: Evaluating test samples

We used an **80-20 train-test split** to evaluate our ResNet model, allocating 80% of the data for training and 20% for testing. This approach balances learning and evaluation, ensuring the model captures patterns effectively while providing an unbiased performance assessment. The results from this split were promising and further verified using a hidden test set to assess generalization on unseen data.

Specifications

Google Colab Environment
- **CPU**: Intel Xeon Processor
- **GPU**: NVIDIA A100
- **Python Version**: Python 3.9
- **System Memory**: 64GB
- **CUDA Version**: 10.2
- **Torch Version**: 2.6

Acknowledgments

We would like to acknowledge the use of OpenAI's GPT-4 in the preparation of this report. GPT-4 provided assistance in refining descriptions for the report.

References

1. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778. https://doi.org/10.1109/CVPR.2016.90

2. Zagoruyko, S., & Komodakis, N. (2016). Wide Residual Networks. *Proceedings of the British Machine Vision Conference (BMVC)*, 87. https://doi.org/10.5244/C.30.87