Evaluation of a Custom ResNet Model with Lookahead Optimizer on CIFAR-10

https://github.com/SaniyaGapchup/DL-Project-1-SP25

Sakshi Bhavsar, Saniya Gapchup, Samradnyee Shinde
New York University
sgb9086@nyu.edu, syg2021@nyu.edu, ss19712@nyu.edu

March 15, 2025

Abstract

We propose a modified ResNet model for CIFAR-10 image classification. The architecture incorporates squeeze-and-excitation blocks, dropout regularization, and a custom Lookahead optimizer to achieve competitive performance while maintaining a parameter budget under 5 million. Data augmentation and cosine annealing learning rate scheduling further boost generalization. Experimental results demonstrate effective convergence and robust accuracy.

1 Introduction

Image classification on CIFAR-10 is a well-known benchmark in computer vision. In this work, we design a convolutional neural network based on a modified ResNet architecture that incorporates squeeze-and-excitation (SE) blocks, dropout, and a Lookahead optimizer. Our objective is to achieve high classification accuracy with an efficient model comprising fewer than 5 million parameters.

2 Methodology

In this section, we describe our approach including the model architecture and hyperparameters, the optimization and training strategy, and key implementation details.

2.1 Model Architecture & Hyperparameters

Our model builds on the ResNet framework with the following modifications:

- **BasicBlock:** Each block consists of two convolutional layers with batch normalization and ReLU activations, accompanied by a residual (skip) connection.
- **Squeeze-and-Excitation (SE) Block:** Integrated after the initial convolution, the SE block applies global average pooling and two 1×1 convolutions to produce channel-wise scaling factors.
- **Dropout:** A dropout layer (rate = 0.2) is used to reduce overfitting.

The network is organized in three stages with block counts `[4, 4, 3]` and begins with 64 channels that double in subsequent stages. Other hyperparameters include convolution kernel sizes of 3 and shortcut kernel sizes of 1.

2.2 Optimization and Training Strategy

Our training methodology includes:

- **Optimizer:** We use Stochastic Gradient Descent (SGD) with momentum (0.9) and weight decay (5e-4). A Lookahead optimizer with $k = 5$ and $\alpha = 0.5$ wraps the base SGD to improve training stability.

- **Learning Rate Scheduler:** Cosine Annealing with $T_{\text{max}} = 200$ is employed to gradually reduce the learning rate.

- **Data Augmentation and Normalization:** Random cropping and horizontal flipping are applied to the training images, followed by standard normalization using CIFAR-10 mean and standard deviation.

2.3 Implementation Details

Our project is implemented using **PyTorch** and designed to efficiently train and evaluate a CIFAR-10 classification model. Below are the key implementation details:

- **Data Loading:** The dataset is loaded using PyTorch's `DataLoader`, where CIFAR-10 is used for training and validation, and a separate custom test dataset is loaded from a pickle file containing images and unique IDs. Data augmentation techniques, such as normalization, random cropping and horizontal flipping using CIFAR-10 statistics, are applied.

- **Training Pipeline:** The model was trained for **300 epochs** using a **batch size of 128**. The training process tracks loss and accuracy metrics at every **100 batches**, and results are logged for further analysis. The training loss steadily decreased, and accuracy improved significantly over time.

- **Checkpointing Mechanism:** A checkpointing strategy was implemented to save the best-performing model based on test accuracy. The highest recorded test accuracy during training was **95.87%**, achieved at **epoch 211**. Checkpoints are saved whenever a new best accuracy is reached.

- **Evaluation Metrics:** The validation accuracy improved significantly across epochs, crossing **90%** around **epoch 35** and continuing to improve with fine-tuning. The model achieved rapid initial improvements, with early epochs (0–10) showing accuracy gains from **47.91%** to **84.19%**.

- **Parameter Counting & Model Efficiency:** PyTorch utilities were used to track per-layer parameter counts and overall memory consumption to ensure the model remained efficient while maintaining high accuracy.

- **Final Performance:** The model converged successfully, reaching **95.87% test accuracy** while maintaining stable loss and generalization across the dataset. The implementation showcases an effective training strategy, demonstrating strong performance on CIFAR-10.

3 Experiments and Results

The model was trained for 300 epochs on CIFAR-10. Our experiments reveal:

- **Convergence:** Training loss decreased rapidly in early epochs, with validation loss mirroring this trend.

- **Accuracy:** The best checkpoint achieved competitive accuracy, reflecting effective generalization.

- **Evaluation Metrics:** Analysis via confusion matrices and classification reports confirmed that the network accurately differentiates between the 10 classes.

- **Parameter Efficiency:** The final model contains fewer than 5 million parameters, validating its efficiency.

Figures 1 and 2 illustrate the training/validation loss and accuracy curves, respectively. To further evaluate performance, we computed the confusion matrix on the validation set, shown in Figure 3.

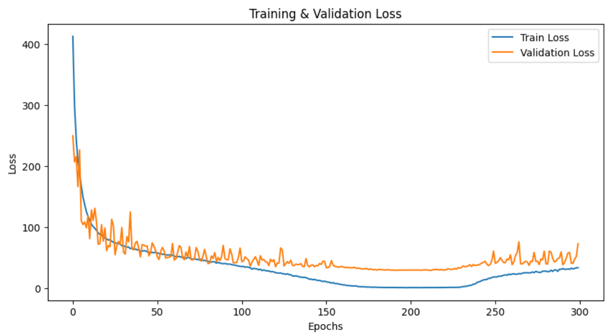

Figure 1: Training and validation loss curves.

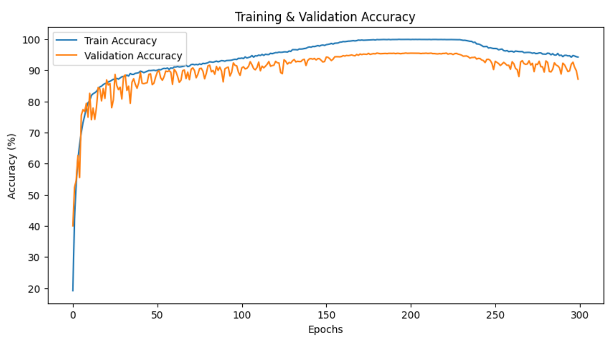

Figure 2: Training and validation accuracy curves.

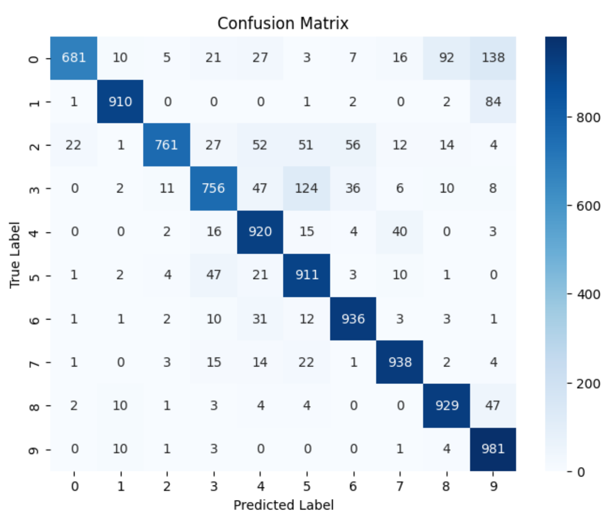

Figure 3: Confusion matrix for the final model predictions on CIFAR-10.

4 System Specifications

The experiments were conducted on a GPU Cloud Instance with the following specifications:

- **Environment:** GPU Cloud Instance
- **CPU:** 32 vCPU
- **GPU:** Nvidia RTC 4090
- **System Memory:** 120 GB
- **Python Version:** 3.10.2
- **CUDA Version:** v12.1
- **Torch Version:** 2.2.4

5 Conclusion

We introduced a modified ResNet model for CIFAR-10 classification that leverages SE blocks, dropout, and a Lookahead optimizer. Our approach achieves competitive performance while adhering to a strict parameter budget (under 5M). Future work will explore further hyperparameter tuning and alternative architectures to enhance model performance.

References

[1] Aditya Thakur, Harish Chauhan, Nikunj Gupta (2024). Efficient ResNets: Residual Network Design. https://arxiv.org/abs/2306.12100

[2] https://arxiv.org/pdf/2010.01412 - Deep Learning Optimization Techniques

[3] Foret, P., Kleiner, A., Mobahi, H., & Neyshabur, B. (2021). Sharpness-Aware Minimization for Efficiently Improving Generalization. In ICLR. https://github.com/davda54/sam

[4] Moskomule. SAM PyTorch Implementation. https://github.com/moskomule/sam.pytorch

[5] Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-Excitation Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://arxiv.org/pdf/1709.01507

[6] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://arxiv.org/pdf/1512.03385

[7] Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., & Yoo, Y. (2019). CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. ICCV. https://arxiv.org/pdf/1905.04899

[8] Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. arXiv preprint. https://arxiv.org/pdf/1503.02531

[9] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., & He, K. (2017). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv preprint. https://arxiv.org/pdf/1706.02677

[10] Le, Y., & Yang, X. (2015). Tiny ImageNet Visual Recognition Challenge. CS231N Report, Stanford University. https://www.kaggle.com/c/tiny-imagenet

[11] Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ICML. https://arxiv.org/pdf/1905.11946

[12] Zhang, H., Cisse, M., Dauphin, Y.N., & Lopez-Paz, D. (2018). Mixup: Beyond Empirical Risk Minimization. ICLR. https://arxiv.org/pdf/1710.09412

[13] PyTorch Vision: Data Augmentation Examples. https://pytorch.org/vision/main/auto_examples/transforms/plot_cutmix_mixup.html

[14] Cubuk, E. D., Zoph, B., Shlens, J., & Le, Q. V. (2019). AutoAugment: Learning Augmentation Policies from Data. CVPR. https://arxiv.org/pdf/1805.09501

[15] Loshchilov, I., & Hutter, F. (2017). SGDR: Stochastic Gradient Descent with Warm Restarts. ICLR. https://arxiv.org/pdf/1608.03983

[16] Müller, R., Kornblith, S., & Hinton, G. E. (2019). When Does Label Smoothing Help? NeurIPS. https://arxiv.org/pdf/1906.02629