Optimizing Modified ResNet Architecture for CIFAR-10 Image Classification with Parameter Constraints

Aarav Pandya (ap7641), Pradhyumn Bhale (pb2777), Vivek Nayak(vgn2004)

Repository

https://github.com/VivekBits2210/deep-learning-mini-project

Objective

This report details the design and assessment of an optimized residual network (ResNet) architecture, specifically developed for achieving maximal test accuracy on the CIFAR-10 image classification dataset while maintaining a maximum parameter limit of 5 million. The study investigates multiple approaches to enhance network performance and efficiency, striking a balance between model complexity and computational resource demands. The findings showcase the success of the proposed architecture in complying with parameter constraints and attaining exceptional classification accuracy on the CIFAR-10 dataset.

Introduction

Residual Networks (ResNets) (He et al. 2016a)(Szegedy et al. 2017) have significantly impacted the deep learning domain by facilitating the training of deeper convolutional networks with skip connections, effectively addressing the vanishing gradient issue. (He et al. 2016b) introduced identity mappings in "Identity Mappings in Deep Residual Networks," which substantially enhanced deep residual networks' performance. Furthermore, (Wightman, Touvron, and Jégou 2021) proposed an improved training procedure for ResNet in "ResNet Strikes Back: An Improved Training Procedure in timm," contributing to the further refinement of ResNet architectures. This report centers on developing an optimized ResNet architecture for the CIFAR-10 image classification dataset (Krizhevsky 2009), while adhering to a maximum parameter limit of 5 million. The primary goal is to achieve the highest possible test accuracy under this constraint. Numerous resources on Github (Liu 2023) provide detailed descriptions of the training process for Resnets from scratch. We investigate some of these optimization techniques to maximize our architecture's training accuracy.

ResNet models primarily consist of residual blocks, implementing the function:

$$ReLU(S(x) + F(x)) \quad (1)$$

where S(x) denotes the skip connection and F(x) represents a block implementing conv $->$ BN $->$ relu $->$ conv $->$ BN, with "BN" referring to batch normalization (Ioffe and Szegedy 2015). By connecting these blocks sequentially, a deep ResNet is constructed.

The design of ResNet architectures requires optimizing various hyperparameters, including the number of channels (C_i) in the $i-th$ layer, the filter size (F_i) in the $i-th$ layer, the kernel size (K_i) in the $i-th$ skip connection, and the pool size (P) in the average pool layer. This report investigates the exploration and adjustment of these hyperparameters to develop an efficient and high-performing ResNet model for the CIFAR-10 dataset, offering insights into the balance between model complexity and computational resources.

Methodology

Model Design

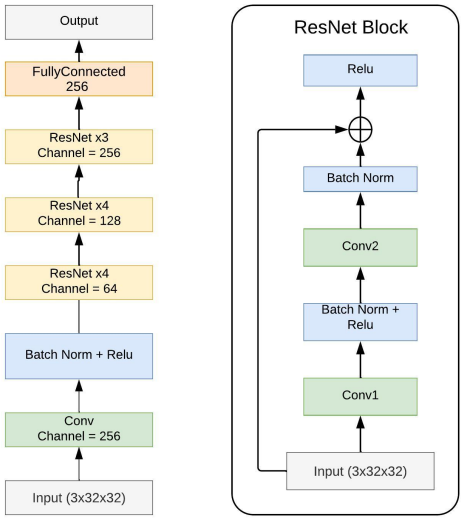

Figure 1: Model design

Our primary focus was on experimenting with different architectural choices to understand their impact on model

performance. The base architecture we used was a simplified ResNet model, which consists of residual blocks known for their ability to effectively learn hierarchical features in deep networks.

We made several modifications to the base architecture to improve its performance.

- **Dropout Layer:** We added a dropout layer(Srivastava et al. 2014) to the model to introduce regularization, which helps prevent overfitting by randomly dropping neurons during training. This encourages the model to learn more robust features by relying on the ensemble of neurons rather than individual ones.
- **Reducing the Number of Blocks:** We experimented with minimizing the number of residual blocks in each layer. The goal was to find the optimal trade-off between model complexity and performance, as a smaller model is computationally more efficient and less prone to overfitting.
- **Optimizers** We tested different optimization algorithms, such as Stochastic Gradient Descent (SGD) and Adam, to find the one that led to better convergence and performance. Our experiments showed that SGD outperformed Adam in our specific scenario.
- **Pruning** We also tried using pruning to reduce the size of the CNN by removing weights or entire neurons that contribute the least to the model's performance. This can make the model faster and more memory-efficient without significantly sacrificing accuracy. However, we weren't able to implement it successfully for it to work and we have commented out the code for it in our Jupyter notebook on Github.

Training Strategy

We trained the models using a GPU-accelerated environment to speed up the process. To find the best model configuration, we increased the number of training epochs to allow the models more opportunities to learn from the dataset. We monitored the validation accuracy to avoid overfitting and to select the model with the best generalization capabilities.

The architecture summary for the model parameters is detailed in Table 1. The hyperparameter values for our best model are given in Table 2.

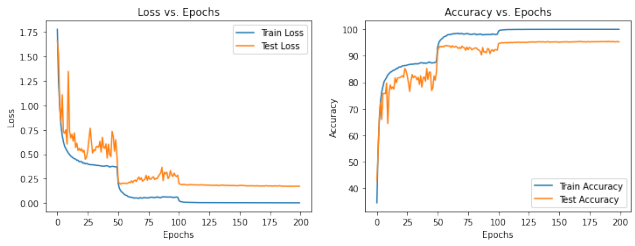

Figure 2: Training and Test Accuracy

Model Architecture

Our model is a simplified ResNet model in PyTorch. The model consists of two primary classes: SimplifiedBasicBlock and SimplifiedResNet.

Table 1: Architecture Summary

Total Parameters	4,697,162
Trainable parameters	4,697,162
Non-trainable parameters	0
Input Size	0.01 MB
Forward/backward pass size	25.88 MB
Params size	17.92 MB
Estimated Total Size	43.81 MB

Table 2: Hyperparameters

Loss	Cross-entropy
Optimizer	SGD
Batch Size	100
Learning Rate Scheduler	MultiStepLR
gamma	0.1
Momentum	0.9
Weight Decay	5e-4
Epochs	200

The SimplifiedBasicBlock class is a custom implementation of a basic residual block, which is the fundamental building block of a ResNet architecture. It inherits from the PyTorch *nn.Module* class and defines the following components:

- Two convolutional layers (Conv2d) with batch normalization (BatchNorm2d) and ReLU activation (ReLU).
- A shortcut connection that either directly connects the input to the output or includes a convolutional layer and batch normalization depending on the stride and channel dimensions.

The forward method in *SimplifiedBasicBlock* defines the computation for a single basic block, which involves applying the convolutional layers and then adding the shortcut connection before applying the final ReLU activation.

The SimplifiedResNet class defines the overall structure of the simplified ResNet model. It also inherits from the PyTorch $nn.Module$ class and includes the following layers:

- A single convolutional layer (Conv2d) with batch normalization (BatchNorm2d) and ReLU activation (ReLU) for the initial feature extraction.
- Three layers of basic blocks, with the number of blocks in each layer specified by the *'num blocks'* parameter.
- An adaptive average pooling layer *(AdaptiveAvgPool2d)* to reduce the spatial dimensions to 1x1.
- A dropout layer *(Dropout)* for regularization.
- A fully connected layer *(Linear)* to produce the final class scores.

The forward method in SimplifiedResNet defines the computation for the entire model, which involves passing the input through the initial convolutional layer, followed by the three layers of basic blocks, adaptive average pooling, dropout, and the fully connected layer to produce the final output.

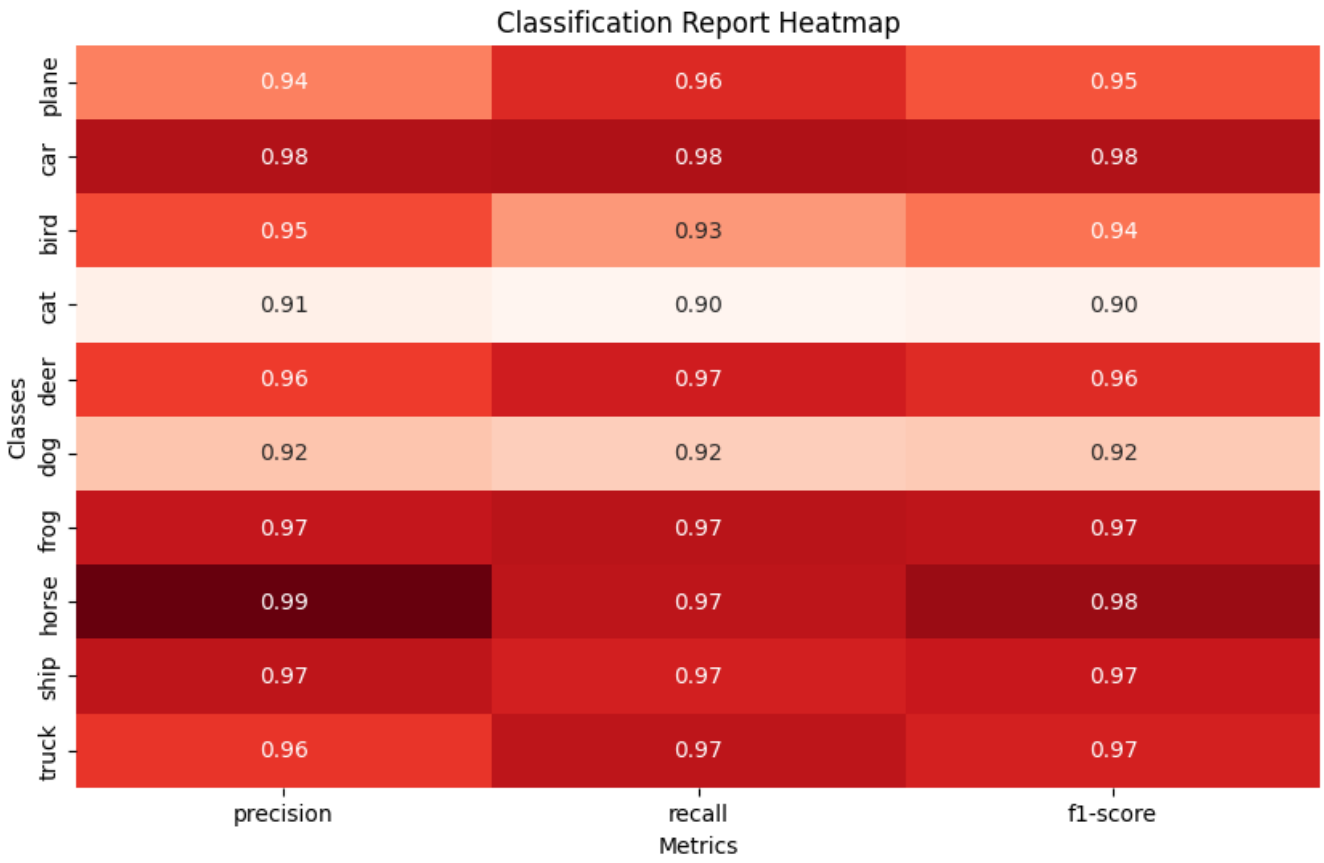

Figure 3: Precision/Recall for each category.

Results

Figure 4: Classification

Our training and test accuracy are tabulated in Table 3. We are able to get a remarkable test accuracy of **95.46 percent** with less than 4.7 mil parameters.

Table 3: Results

Train loss	0.0027
Train Acc	99.9980
Test Loss	0.1718
Test Acc	95.46

We present our training and testing loss and accuracy visualizations in Fig. 2. Fig. 4 displays the classification results of five randomly selected samples. Additionally, we provide a per-class precision/recall metric in Fig. 3. Our model accurately predicts every class, except for the Cat and Dog classes, with high precision and recall.

Conclusion

Throughout the design and training process, we learned several valuable lessons:

- **Regularization:** The addition of the dropout layer proved to be an effective method for reducing overfitting and improving model generalization.
- **Model Complexity:** Reducing the number of residual blocks highlighted the importance of balancing model complexity and performance. Simplifying the model can sometimes lead to similar or even better performance with reduced computational overhead.
- **Optimizer Selection:** The choice of the optimization algorithm played a crucial role in the convergence and final performance of the models. Our experiments demonstrated that the choice of optimizer is problem-dependent, and it is essential to experiment with different optimizers to find the one best suited for the task at hand.

In conclusion, our methodology allowed us to iteratively experiment with different architectural choices and training strategies, leading to valuable insights into the design of

deep convolutional neural networks for image classification tasks.

References

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016a. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016b. Identity Mappings in Deep Residual Networks. In Leibe, B.; Matas, J.; Sebe, N.; and Welling, M., eds., *Computer Vision – ECCV 2016*, 630–645. Cham: Springer International Publishing. ISBN 978-3-319-46493-0.

Ioffe, S.; and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, abs/1502.03167.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images.

Liu, K. 2023. Train CIFAR10 with PyTorch (https://www.github.com/kuangliu/pytorch-cifar).

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56): 1929–1958.

Szegedy, C.; Ioffe, S.; Vanhoucke, V.; and Alemi, A. A. 2017. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, 4278–4284. AAAI Press.

Wightman, R.; Touvron, H.; and Jégou, H. 2021. ResNet strikes back: An improved training procedure in timm. arXiv:2110.00476.