Deep Learning - ResNet under 5 million Parameters

Sampreeth Avvari* **Barath Rama Shankar*** **Dhruv Sridhar***
`spa9659@nyu.edu, br2543@nyu.edu, ds7395@nyu.edu`
New York University

Abstract

In this project, we developed a customized Residual Network (ResNet)[2] tailored for the CIFAR-10[1] image classification challenge. Our goal was to optimize test accuracy while adhering to a strict parameter limit of 5 million trainable parameters, ensuring the model's suitability for storage on devices with limited capacity, such as IoT or edge devices. Our redesigned ResNet model contains fewer than 5 million parameters, yet it achieves a test accuracy of 97.12% on the CIFAR-10 dataset. This performance significantly surpasses that of the traditional ResNet18 model, which contains over 11 million parameters. We attribute our model's enhanced performance to a combination of strategic training approaches and optimal ResNet hyperparameters. The models and associated code are accessible via the **GitHub repository[3]**.

Introduction

ResNet stands as a leading deep neural network widely employed for various computer vision tasks. The prevailing approach in network design has been to increase complexity and depth to enhance accuracy. However, these enhancements often do not translate into improvements in network efficiency in terms of size and processing speed. This limitation is critical in applications like embedded systems, robotics, self-driving vehicles, and augmented reality, where quick data processing is crucial on devices with limited computational power. To address this need for more memory-efficient models that still deliver robust performance, developments like MobileNet and WideNet have been pursued. This article outlines our efforts to develop a more compact version of the ResNet-18 model. We will detail our approach, adjustments made to the model's parameters, and the optimization techniques employed to achieve an optimal configuration. This new setup maintains less than 50% of the original trainable parameters while achieving higher accuracy than the standard ResNet-18 model.

Methodology

The primary objective of this study is to develop a model constrained to fewer than 5 million trainable parameters.

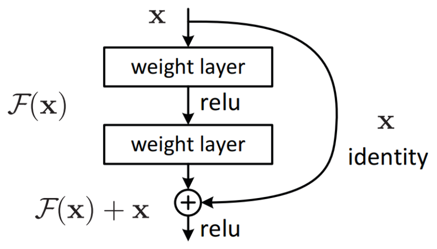

Figure 1: ResNet Identity Mapping [2]

The number of trainable parameters is predominantly influenced by the number of residual blocks, layers, and channels in each Conv2D layer. Additional variations explored include different optimizers and regularizers. The various configurations tested and their respective evaluations are presented in Table 1.

The hyper parameters that affect the number of trainable parameters and that can be changed to improvise a model's performance are

$$\begin{aligned}
N &:= \text{Total count of distinct layers}\\
B_i &:= \text{Quantity of residual blocks in the } i\text{th residual layer}\\
C_i &:= \text{Total number of channels in the } i\text{th residual layer}\\
F_i &:= \text{Size of convolution kernel in the } i\text{th residual layer}\\
K_i &:= \text{Dimension of skip kernel in the } i\text{th residual layer}\\
P &:= \text{Size of kernel used for average pooling operations}
\end{aligned}$$

Impact of Data Augmentation on Performance

Data augmentation is a critical technique in enhancing the generalization of deep learning models, particularly in the context of image recognition tasks and avoid overfitting.

In our experiments with ResNet, various combinations of data augmentation techniques were employed to assess their impact on model accuracy. Initially, basic augmentations such as flipping and cropping were used, which provided a foundational improvement in model robustness as reflected by a baseline accuracy increase. Subsequently, more complex augmentations including auto augmentation and perspective transformations were integrated, aiming to introduce more variability and challenge to the model. The incorporation of auto augmentation alongside flipping and cropping significantly enhanced the accuracy, indicating the model's improved ability to generalize from more complex augmented data.

Table 1: Best accuracies achieved by different augmentation combinations.

Augmentation Techniques	**Best Accuracy (%)**
Flip, Crop	93.76%
Auto, Flip, Crop	95.34%
Auto, Flip, Crop, Perspective	**97.12 %**

Average Pooling P

Pooling layers play a crucial role in convolutional neural networks (CNNs) by reducing the spatial dimensions of the input feature maps. This operation not only decreases the computational load and the complexity of the model but also helps in making the representation more manageable and robust to variations in the input data.

There are several types of pooling operations commonly used in CNNs:

- **Max Pooling**: Selects the maximum element from the region of the feature map covered by the filter. This method is particularly effective at capturing the presence of specific features in the input regions.
- **Average Pooling**: Calculates the average of elements in the region of the feature map covered by the filter, which helps in smoothing the feature responses and preserving background information.
- **Global Pooling**: Reduces each channel of the feature map to a single summary statistic. A global average pooling calculates the average of all elements in the feature map, typically used before the final classification layer.

In ResNet architectures, pooling layers are strategically incorporated to reduce feature map dimensions progressively, allowing the network to increase the depth without a significant increase in computational cost. Specifically, average pooling is often preferred in later stages of ResNet models due to its ability to retain more general information compared to max pooling. This is beneficial for tasks where contextual information from larger receptive fields is crucial.

The integration of average pooling layers in ResNets is described as follows:

$$P_{avg}(F_{ij}) = \frac{1}{K \times K} \sum_{k=1}^{K} \sum_{l=1}^{K} F_{i+k,j+l} \qquad (1)$$

Here, P_{avg} represents the average pooling operation, F_{ij} is the input feature map at position (i, j), and K is the size of the kernel used for pooling. This operation is applied after certain convolutional layers to reduce the dimensionality of feature maps, thus ensuring that deeper layers focus on increasingly abstract and global representations of the input data.

The use of pooling in ResNets aids in maintaining a manageable number of parameters, which is critical for training deep networks efficiently and effectively. Moreover, by reducing overfitting, pooling layers enhance the generalization ability of ResNet models across diverse datasets and challenging recognition tasks.

Residual Layers N, and Residual block B_i

In our methodology, we investigated the impact of varying the number of residual layers and blocks within each layer on the model's performance. Residual layers are fundamental components of ResNet architectures that help alleviate the vanishing gradient problem in deep networks by allowing activations to flow through a shortcut connection that skips one or more layers. Each residual layer is composed of multiple residual blocks, which are the basic unit of composition in these networks, consisting primarily of convolutional layers.

The number of these layers and blocks directly influences the model's capacity and computational complexity. Increasing the number of layers typically allows the network to learn more complex features at different levels of abstraction, enhancing its ability to perform on more challenging tasks. Similarly, more blocks within a layer can provide finer representations but at the cost of increased trainable parameters.

To quantify these effects, we systematically varied both the number of residual layers and the number of blocks within those layers. This approach allowed us to explore a spectrum of architectures, balancing between computational efficiency and predictive accuracy. The configurations tested ranged from shallow networks with fewer blocks to deeper models with multiple blocks, aiming to optimize the network structure under the constraint of 5 million trainable parameters. Our findings, as detailed in Table 1, demonstrate the trade-offs between depth, complexity, and performance, providing critical insights into the scalable design of ResNet architectures for efficient deep learning applications.

Table 2: Number of Blocks vs. Accuracy

Number of Blocks in each layer	**Accuracy**
3, 3, 3, 3	**97.12%**
3, 3, 2, 3	96.84%
3, 4, 6, 3	91%
2, 2, 2, 2	90%

Number of Channels in Residual Layer C_i

Channels in convolutional neural networks represent the number of feature maps producedby each layer, which capture various features of the input at different levels of abstraction. we assessed how the number of channels in each residual layer affects model performance and efficiency. Increasing the number of channels can enhance model accuracy by capturing more complex features, but it also raises the model's parameter count, computational complexity, and memory usage. Balancing these factors, we experimented with different channel sizes while keeping the total number of trainable parameters under five million. This allowed us to explore the trade-offs between computational efficiency and performance. Our varied configurations, aimed at optimizing both low-complexity environments and high-accuracy needs, are detailed in Table 2. These results underscore the significant impact channel size has on designing efficient deep learning models, especially under resource constraints.

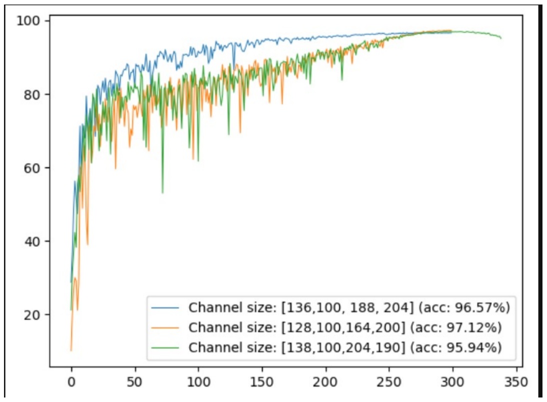

Figure 2: Channel Sizes Vs Accuracy

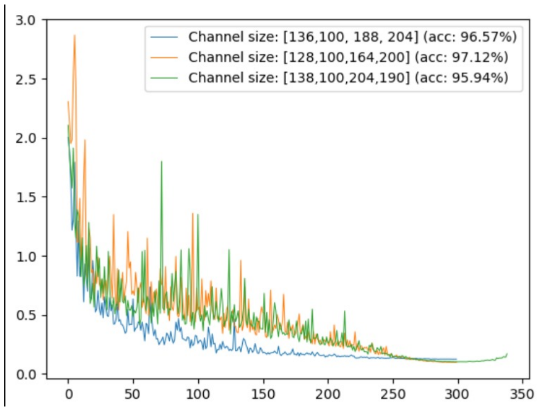

Figure 3: Channel Sizes Vs Test Loss

Table 3: Channel Configurations and Accuracy

Channels	Accuracy
128, 100, 160, 200	**97.12%**
136, 100, 188, 204	91%
32, 64, 128, 256	91%
16, 32, 64, 128	79%

Optimizer and Scheduler

we evaluated various optimizers and learning rate schedulers to enhance our ResNet model's performance within the parameter constraint. We utilized Stochastic Gradient Descent (SGD) with momentum for its generalization, AdaDelta for its adaptive learning rates, and AdamW for incorporating weight decay, which optimizes L2 regularization.

For learning rate schedulers, StepLR and CosineAnnealingLR were chosen. StepLR decreases the learning rate at predetermined intervals, helping overcome plateaus in the loss landscape, while CosineAnnealingLR adjusts the rate in a cosine pattern, promoting exploration in the early stages and stabilization towards the end of training.

Cosine Annealing Learning Rate (CosineAnnealingLR) is a strategy that adjusts the learning rate following a periodic cosine function. The primary idea behind this approach is to start with a larger learning rate ($\eta_{\max}$) that allows faster convergence and then gradually reduce it to a lower boundary ($\eta_{\min}$) over time. This reduction process follows a cosine curve, which allows the learning rate to "anneal" or cool down slowly, avoiding abrupt changes that could disrupt the learning process.

The specific equation used to calculate the learning rate (η_t) at any given epoch t is:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_{\max}}\pi\right)\right)$$

Here, η_t is the adjusted learning rate at time t, $\eta_{\min}$ and $\eta_{\max}$ are the minimum and maximum bounds of the learning rate, respectively. T_{cur} is the current epoch number, and $T_{\max}$ is the maximum number of epochs over which the schedule operates.

The use of the cosine function ensures a smooth transition of the learning rate from $\eta_{\max}$ to $\eta_{\min}$, emulating a cosine wave's crest and trough. This strategy has been shown to facilitate the convergence to a better local minimum by periodically increasing the learning rate, potentially allowing the model to escape local minima.

The selection of these tools was instrumental in refining our model's training process, ensuring efficient convergence and achieving a balance between exploration and exploitation in the parameter search space. The performance of each optimizer and scheduler is in Table-4.

Table 4: Optimizer and Scheduler Combinations vs. Accuracy

Optimizer & Scheduler	Accuracy
SGD(momentum) + CosineAnnealingLR	**97.12%**
AdaDelta + CosineAnnealingLR	96.04%
AdamW + CosineAnnealingLR	96.57%
SGD + StepLR	92%

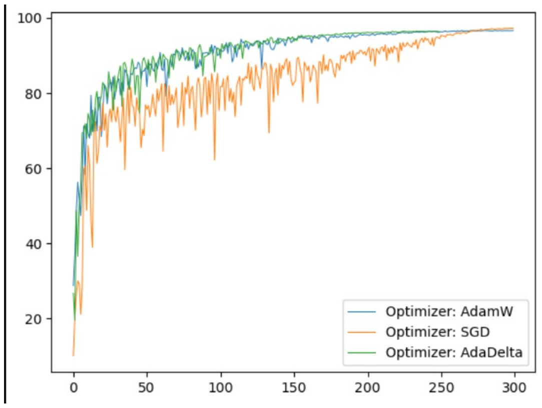

Figure 4: Optimizer Vs Accuracy

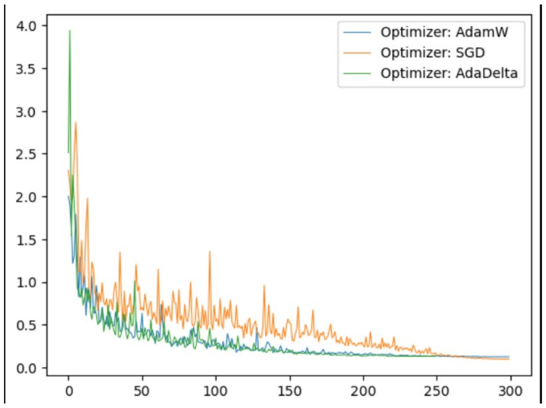

Figure 5: Optimizer Vs Test Loss

Reguralizer

In our streamlined ResNet architecture, we incorporated batch normalization which served a dual purpose: it not only stabilized the learning process by normalizing layer inputs but also introduced a regularization effect. Regularization techniques, such as batch normalization, help prevent overfitting by discouraging complex models during training. This ensures the model performs well on unseen data, enhancing its ability to generalize. The integration of batch normalization was pivotal, contributing to the robustness and maintaining high accuracy in our model's predictions.

$$\mathrm{BN}(x_i) = \gamma \left(\frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \beta$$

where:

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{(mini-batch mean)}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \quad \text{(mini-batch variance)}$$

$$\gamma = \text{scale parameter (learnable)}$$

$$\beta = \text{shift parameter (learnable)}$$

$$\epsilon = \text{a small constant for numerical stability}$$

Convolution Kernel Size F_i, Skip Connection Kernel K_i

In the ResNet architecture, the kernel size F of the convolutional layers is fixed at 3×3. This configuration offers an optimal trade-off between the model's ability to capture spatial dependencies and its computational efficiency. By choosing a small kernel size, we also minimize the number of parameters, thereby making the network less prone to overfitting and quicker to train.

The skip connection kernel size K, set to 1, contributes to the compactness of the model. It acts as a feature selector and preserves essential information, allowing the network to remain deep without a significant increase in computational requirements.

These kernel size choices of F and K are instrumental in maintaining the delicate balance between model depth and parameter count, enabling our ResNet models to achieve high accuracy while ensuring computational efficiency.

Results

In this section, we present a comparative analysis of the learning and accuracy curves for our optimally performing models, along with a detailed discussion of their hyperparameters. As illustrated in Figure 2, our best-performing model, which is subject to a parameter constraint of fewer than 5 million, attains an accuracy of 97.12%. This significantly surpasses the accuracy of the original ResNet model, which has a higher parameter count but only reaches an accuracy of 90%.

Table 5: ResNet Configuration and Performance Analysis

SlNo.	Architecture	Augmentation	Channels	Optimizer+Scheduler	Epochs	No of Blocks	No. of Parameters	Test Accuracy %
1	**Res24**	**Auto, Flip, Crop**	**128,100,164,200**	**SGD+m, Cosineannealing**	**300**	**3,3,3,3**	**4.9M**	**97.12**
2	**Res24**	**Auto, Flip, Crop, Perspective**	**138, 100, 204, 190**	**Adadelta, Cosineannealing**	**250**	**3,3,2,3**	**4.9M**	**96.4**
3	Res24	Auto, Flip, Crop, Perspective	138,100, 204, 190	Adadelta, Cosineannealing	200	3,3,2,3	4.9M	96.18
4	Res24	Auto, Flip, Crop	94,192,120,200	SGD+m, Cosineannealing	200	3,3,2,3	4.9M	95.94
5	Res34	Flip, Crop	32, 64, 128, 256	SGD+m, StepLR	150	3,4,6,3	5.3M	91
6	Res22	Flip, Crop	64, 128, 256	SGD+m, StepLR	100	3, 4, 3	4.6M	89
7	Res34	Flip, Crop	64, 128, 128, 256	SGD+m, StepLR	100	3,4,6,3	11.7M	90
8	Res18	Flip, Crop	16, 32 , 64, 128	SGD+m, StepLR	100	2,2,2,2	701466	79
9	**Res24**	**Auto, Flip, Crop, Perspective**	**138,100,204,190**	**SGD+m, Cosineannealing**	**300**	**3,3,2,3**	**4.9M**	**96.82**

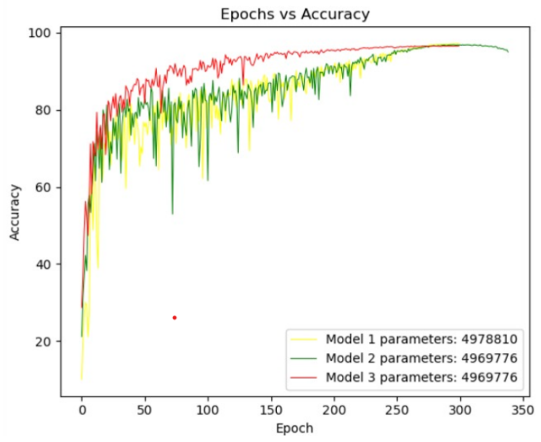

Figure 6: Accuracy Comparison of best Models

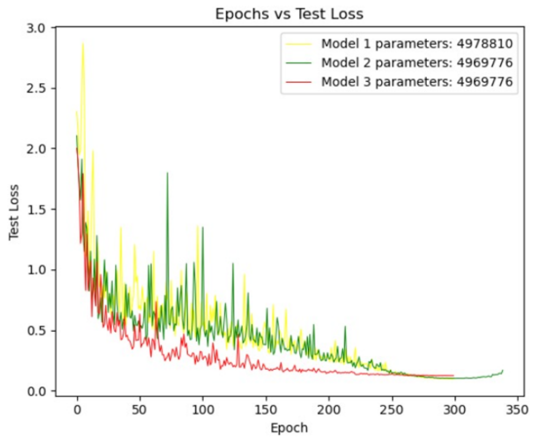

Figure 7: Test Loss Comparison of best Models

Conclusion

In our research, we explored a variety of hyperparameter optimizations to maximize the accuracy of a ResNet model constrained to fewer than 5 million parameters. This systematic tuning process enabled us to develop a model that achieved a test accuracy of 97.12% . This result underscores the potential of strategic hyperparameter adjustments in enhancing model performance within parameter limits.

References

[1] Alex Krizhevsky. *CIFAR-10 and CIFAR-100 datasets*. University of Toronto. Available at: https://www.cs.toronto.edu/~kriz/cifar.html. Accessed: 2024-04-12.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*, arXiv preprint arXiv:1512.03385. 2015. Available at: https://arxiv.org/abs/1512.03385.

[3] ds28-ops. *DL_Resnet GitHub Repository*. GitHub. Available at: https://github.com/ds28-ops/DL_Resnet. Accessed: 2024-04-12.

[4] Nikunj Gupta, *Efficient_ResNets*, GitHub repository, Available at: https://github.com/Nikunj-Gupta/Efficient_ResNets.

[5] Anant Singh, *NYU-ResNet-On-Steroids*, GitHub repository, Available at: https://github.com/95anantsingh/NYU-ResNet-On-Steroids.

[6] Authors, *When Vision Transformers Outperform ResNets*. 2021, URL: https://paperswithcode.com/paper/when-vision-transformers-outperform-resnets.