Modified Residual Network
CIFAR-10 Image Classification

Alex Gonzalez, Chloe Kim, Richard Zhong

New York University, New York, USA

https://github.com/Adgonzalez2018/DL_project_1

Abstract

This project aims to develop a modified residual network (ResNet) architecture optimized to achieve the highest test accuracy on a held-out portion of a version of the CIFAR-10 image classification dataset, while adhering to the constraint that the model contains no more than 5 million parameters. To achieve this, we modified the Wide ResNet model, extensively experimenting with the depth and width factor of our model. The adapted model with parameter constraints achieved an accuracy of 82.52% The findings contribute to the ongoing exploration of deep learning architecture design and performance, especially for resource constrained image classification.

Introduction

Residual Networks (ResNets) have fundamentally transformed deep learning by enabling the training of extremely deep architectures through the use of residual blocks and skip connections. These innovations have led to state-of-the-art performance across a wide range of computer vision tasks, from image classification to object detection. However, as ResNets have scaled to thousands of layers, they have faced a critical challenge: diminishing feature reuse. This issue arises because gradients can propagate through the network without being forced to learn meaningful representations in many layers, resulting in inefficient training and slower convergence. While increasing depth has been the primary strategy for improving ResNets, the diminishing returns and computational costs associated with extreme depth have prompted researchers to explore alternative approaches.

Building on the success of ResNets, Wide Residual Networks (WideResNets) were introduced as a novel architectural paradigm that shifts the focus from depth to width. By increasing the number of feature maps within residual blocks while reducing the overall depth of the network, WideResNets address the problem of diminishing feature reuse and significantly improve training efficiency. WideResNets demonstrated that the representational power of residual networks lies not only in their depth but also in the capacity of their individual blocks. Additionally, WideResNets introduced a novel application of dropout within residual blocks, inserting it between convolutional layers rather than on the identity mapping path, which further enhanced regularization and performance.

In this work, we aim to leverage the efficient representation capabilities of WideResNets to design a model that operates within a strict parameter limit of 5 million parameters while achieving superior representational power compared to an equivalent model based on traditional ResNets. Our hypothesis is that by strategically widening the residual blocks and optimizing the balance between depth and width, we can create a compact yet highly expressive architecture that maximizes performance under the given constraints. This approach not only addresses the limitations of traditional ResNets in parameter-constrained settings but also builds on the strengths of WideResNets to deliver a model that is both efficient and effective.

ResNet-18

Introduced by He et al. in 2015, residual network (ResNet) architecture was designed to address vanishing gradients in deep networks. ResNet is any convolutional network with skip connections that bypass one or more layers, which help preserve gradient flow during backpropagation. It is effective for image classification and other computer vision applications.

For ResNet-18, the fundamental basic block of a residual block implements two 3x3 convolutional layers with batch normalization and a ReLu activation.

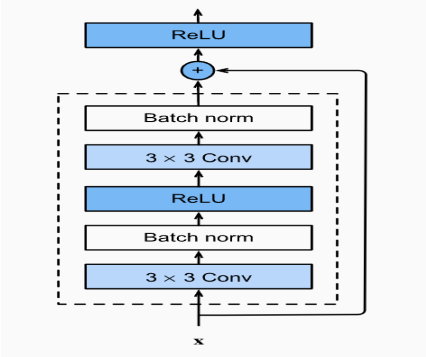

Figure 1: Basic Residual Block

The first convolution layer utilizes a 3x3 kernel with a specified stride, while the second layer preserves the input size. Batch normalization stabilizes training, while skip connection aids the flow of gradients by bypassing the intermediate layers. Residual connections directly add input to outputs, aiding training and enhancing performance. This design improves computational efficiency and optimizes feature extraction by simplifying the network structure and minimizing computational demands.

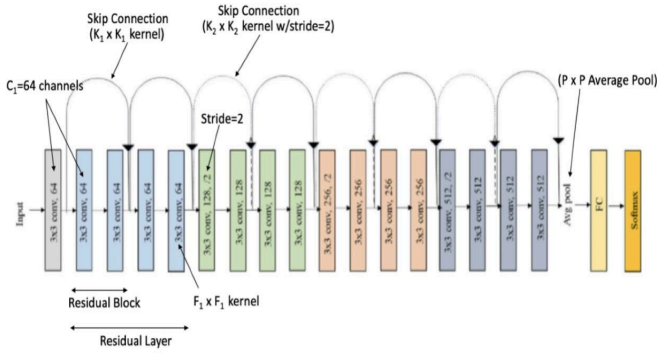

Figure 2 : ResNet-18 Model

The residual layers comprise multiple basic blocks, forming the core ResNet architecture. The stride parameter regulates the downsampling of feature maps within each layer, allowing feature extraction at multiple scales. A Larger stride results in a smaller output feature map, while a smaller stride moves the filter gradually, which preserves more detail in the output. The initial convolutional layers of ResNet-18 have strides set to 2 for efficient downsampling. Skip connections mitigate the vanishing gradient problem, ensuring effective gradient flow during training. The performance of classification is improved with the arrangement of residual layers this way because it enables the network to learn progressively more abstract representations of the input data.

Wide ResNet

The Wide ResNet architecture is a modification of the traditional ResNet architecture that focuses on increasing the width of the network while reducing the depth. In order to increase the width of the network we must use wider residual blocks, where each block uses more and more filters compared to ResNet-18. In our implementation of the Wide ResNet we experimented with models with widths 5 times wider than ResNet-18. Widening the residual blocks allows the network to increase the number of filters per layer while minimizing the total number of parameters. This increase in filters enabled the model to capture more complex features with fewer layers. Optimizing both performance and computational efficiency

Methodology

Data Preprocessing and Augmentation

To enhance generalization, we applied a series of transformations to the training and validation datasets. For the training set, we used the following data augmentation techniques:

1. **Random Crop**: Images were randomly cropped to their original size after padding each side by 4 pixels using the `reflect` mode to preserve edge information.
2. **Random Horizontal Flip**: Images were flipped horizontally with a probability of 0.5 to introduce variability.
3. **Normalization:** Each RGB channel was normalized using the mean `[0.4914, 0.4822, 0.4465]` and standard deviation `[0.2023, 0.1994, 0.2010]`.

For the validation set, only normalization was applied to ensure fair evaluation without augmentation.

Model Architecture and Training Process

We began with an initial WideResNet architecture of depth of 10 and width factor of 10, resulting in 4,764,634 parameters. However, this model exhibited severe overfitting, achieving high training accuracy but poor validation performance. We hypothesized that the wide and shallow layers were overparameterized for the dataset.

To address this, we iteratively adjusted the depth and width while maintaining the parameter limit.

Hyperparameter Tuning

We conducted extensive experiments to determine the optimal hyperparameters for training. The final hyperparameters were as follows:

- **Dropout**: A dropout rate of 0.4 was applied between convolutional layers to regularize the wide residual blocks and prevent overfitting.
- **Optimizers**: We trained two versions of each model: one using SGD with Nesterov momentum and the other using Adam. For SGD, we used a learning rate of 0.01, while for Adam, we used a learning rate of 0.001.
- **Weight Decay**: A weight decay of 5e-3 was applied to both optimizers to regularize the model weights.
- **Learning Rate Scheduler:**: For SGD, we used a multi-step learning rate scheduler with milestones at epochs `[20, 30, 40, 50, 60, 65, 70, 75, 80, 85,

90, 95]` and a multiplicative factor of 0.3. For Adam, we used cosine annealing to smoothly decay the learning rate over the training period.

- **Epochs**: Each model was trained for 100 epochs with a batch size of 128 and a momentum of 0.9 for SGD.

Results and Discussion

We first tested a model with depth of 16 and width of 5, totaling 4,289,754 parameters. While this model performed well on the training and validation sets, it achieved a low accuracy of 23.42% on the Kaggle test set, indicating insufficient generalization.

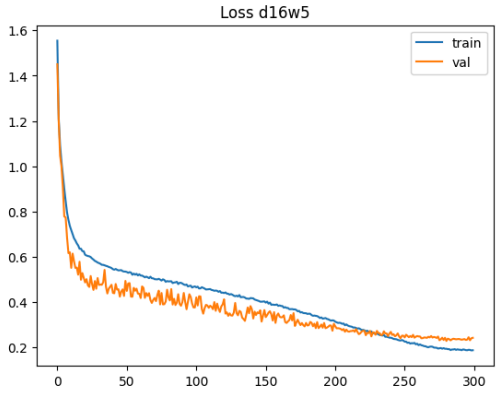

Figure 3 : Loss of Model with Depth: 16, Width: 5

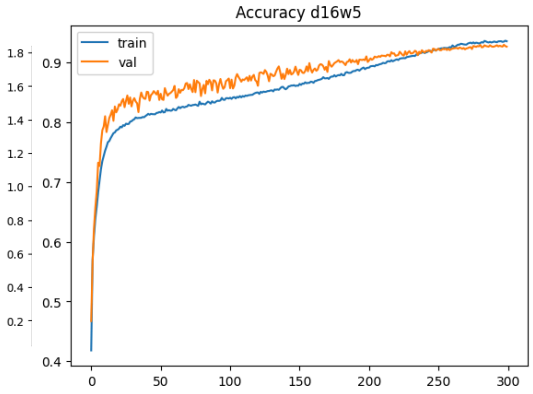

Figure 4: Accuracy of Model with Depth: 15, Width: 5

We experimented with a depth of 22 and width of 5, which had 2,421,786 parameters. This model achieved the best test set accuracy of 82.52%, despite having approximately half the parameters of the previous models. This result confirmed that the earlier models were overparameterized for the dataset size.

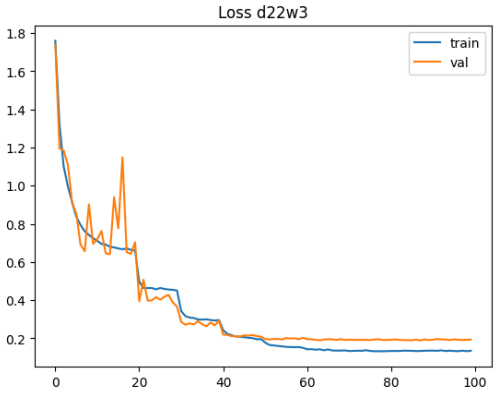

Figure 5: Loss of Model with Depth: 22, Width: 3

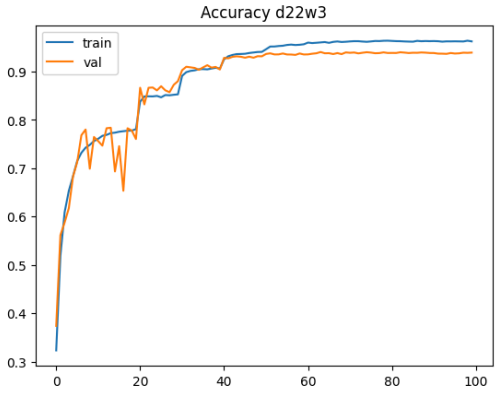

Figure 6: Accuracy of Model with Depth: 22, Width: 3

To further explore the depth-width trade-off, we tested a depth of 28 and width of 3, with 3,294,042 parameters. However, this model performed slightly worse on the test set, achieving an accuracy of 81.86%. Interestingly, we found that training this model first with SGD with Nesterov momentum for 100 epochs, followed by an additional 100 epochs with Adam, improved the test accuracy to 82.12%. This suggests that SGD with Nesterov momentum helps the model reach a reasonable loss landscape, after which Adam can fine-tune the model more effectively.

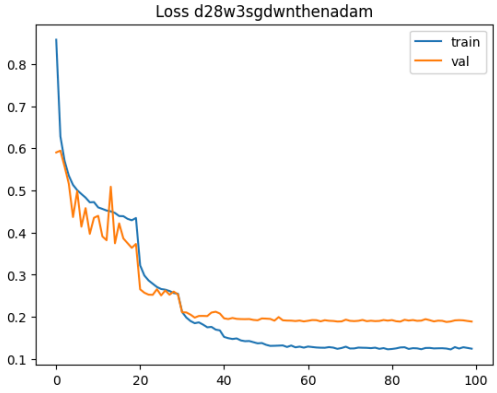

Figure 7: Loss of Model with Depth: 28, Width: 3

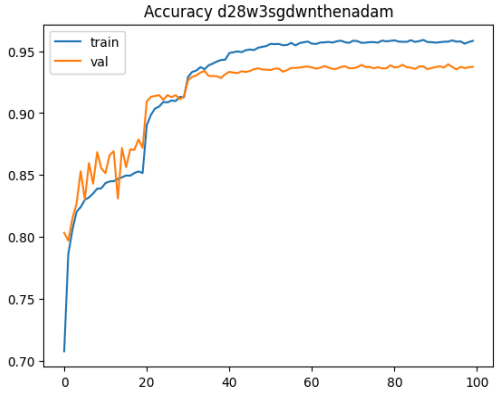

Figure 8 : Accuracy of Model with Depth 28, Width: 3

Models	Parameters	Test Accuracy
Depth: 16, Width: 5	4,289,754	23.42%
Depth: 22, Width: 3	**2,421,786**	**82.52%**
Depth 28, Width: 3	3,294,042	82.12%

Our experiments demonstrated that the model with a depth of 22 and width of 3, comprising 2,421,786 parameters achieved the best balance between representational power and generalization, outperforming both deeper and wider models within the 5 million parameter limit. Additionally, we found that combining SGD with Nesterov momentum and Adam optimization further improved performance, highlighting the importance of optimizer choice and training strategy.

By carefully balancing depth, width, and regularization, we showed that WideResNets can be effectively adapted to parameter-constrained settings, providing a robust framework for designing efficient and high-performing deep learning models.

Acknowledgment

We acknowledge the use of OpenAI's ChatGPT for certain sections of the report.

References

Dive into Deep Learning. 7.6 Residual Networks (ResNet). https://d2l.ai/chapter_convolutional-modern/resnet.html. Accessed: 2025-03-05.

Xternalz. (2017). *WideResNet-pytorch*. GitHub repository. https://github.com/xternalz/WideResNet-pytorch. Accessed: 2025-03-06

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. arXiv. https://arxiv.org/abs/1512.03385. Accessed: 2025-03-06.

Zagoruyko, S., & Komodakis, N. (2016). Wide Residual Networks. arXiv. https://arxiv.org/abs/1605.07146. Accessed: 2025-03-06.