ResNet for CIFAR-10 Image Classification

Kaustubh Agarwal[1], Srujana Kanchisamudram Seshagiribabu[2], Aashir Saroya[3]

New York University[1,2,3]
ka3210@nyu.edu, sk11115@nyu.edu, as17888@nyu.edu

Link to Github repository: https://github.com/kaustubhagarwal/DL-Mini-Project-NYU-24

Abstract

In this project, we designed a ResNet architecture trained on the CIFAR-10 dataset, aiming for optimal test accuracy with fewer than five million parameters. Our findings underscore the significance of Batch Normalization for effective training of deep networks. We evaluated various optimizers in conjunction with batch normalization and investigated the influence of hyperparameters. Data augmentation and normalization techniques were also applied to broaden our training dataset. The resulting model is both computationally efficient and memory-conservative.

Introduction [1]

A ResNet, or Residual Network, addresses the challenge of training very deep neural networks through the implementation of skip connections, which facilitate smoother gradient flow during backpropagation. This allows for the training of networks with hundreds of layers. By selectively bypassing certain layers, ResNet effectively mitigates the vanishing gradient problem and improves information flow within the network. This architectural innovation significantly boosts the performance and trainability of deep neural networks, cementing ResNet's importance in the deep learning field. In contrast, networks without these residual connections are termed plain networks. Additionally, models that integrate multiple parallel skip connections are known as DenseNets.

The formulation F(x) + x in feed-forward neural networks is realized through "shortcut connections," which allow bypassing one or more layers, as depicted in Figure 1. These shortcut connections, often referred to as skip connections, typically perform identity mapping—adding their outputs directly to those of subsequent layers without increasing parameter count or computational complexity. This setup enables straightforward end-to-end training with standard backpropagation and is compatible with widely used libraries without the need for solver modifications.

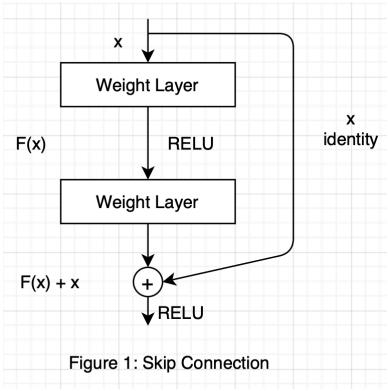


Figure 1: Skip Connection

In this project, we tackled the classification of the CIFAR-10 dataset using a model capped at 5 million trainable parameters. The focus was on evaluating various architectures, selecting effective data augmentation methods, and fine-tuning hyperparameters to optimize test accuracy.

Methodology

Making the Data Loader

The Data Loader is essential in Deep Learning, managing data transfer from diverse sources to a central system. It often involves segmentation into mini-batches and GPU transfers. Our research investigated the impact of different mini-batch sizes on ResNet performance, challenging the assumption that smaller batches always shorten training times.

To enhance our model, advanced data augmentation techniques were implemented for the CIFAR-10 dataset, consisting of 32x32x3 images. Initially, random cropping and horizontal flipping were used, followed by image normalization to adjust pixel intensity values, notably

[1]

boosting model accuracy. Additionally, PyTorch's experimental auto-augmentation policy for CIFAR-10, featuring techniques like blurring and brightness correction, was adopted.
The dataset was divided into training and testing sets using the official CIFAR-10 datasets, with the test set undergoing image normalization to ensure unbiased analysis.

Building the model

In classical machine learning, hyperparameters significantly influence the training process of neural networks. Given the constraints of our project, we meticulously adjusted these hyperparameters to ensure the classification model remained within the parameter limit.

Certain hyperparameters have a more pronounced effect on the accuracy of the classification network. Listed in order of impact, from most to least influential, these include:

1. Number of Residual Blocks in each Residual layer:
The original ResNet paper demonstrated that increasing the number of Residual blocks per layer enhances model performance. We tested various configurations of Residual blocks to assess their impact on CIFAR-10 classification.

2. Number of channels in each Residual layer:
In the Deep Learning community, it's understood that more channels in the initial layer can extract more data from images. We maintained 64 channels in the first layer and varied the increase in channels in subsequent Residual layers to improve classification accuracy.

3. Type of block in each Residual layer:
The ResNet paper introduced the Bottleneck Residual block, which uses 1x1 convolutions to create a bottleneck and reduce parameter count, as an alternative to the Basic Residual block. We analyzed how this design affects the overall accuracy of our trained network.

4. Convolutional Kernel Size in Residual Layer:
Although the impact of this hyperparameter was initially unclear, we conducted tests to better understand its effect on the classification problem.

5. Skip connection kernel size in Residual layer:
Despite testing, we found that this Hyperparameter had minimal influence on the final accuracy.

6. Average Pooling Kernel Size:
The average pooling operation is essential for high accuracy, and we chose to keep its kernel size unchanged to avoid an increase in false positives in the trained network..

Testing different optimizer and schedulers

Different optimization strategies adjust neural network parameters like weights and learning rates to minimize training loss, enhancing model accuracy. During training, loss from each batch is collected, and model weights are adjusted accordingly. Each optimizer employs a unique method for updating weights based on the loss function. Common optimizers used to identify the best-performing model include:

Stochastic Gradient Descent (SGD) with momentum and weight decay:

This is an improved version of the basic SGD algorithm. While vanilla SGD updates weights by subtracting the product of the learning rate (α) and the gradient of the loss ($\nabla L(W)$) from the current weights (W), SGD with momentum utilizes a moving average of gradient descents to update weights, which helps in mitigating the noise in weight updates due to variance among batches. Additionally, to counteract the influence of certain weights on the output, a weight decay factor (λ) is introduced.

The updated weight equation for SGD with momentum and weight decay includes both the momentum term and the weight decay term:

$$v_t = \beta \cdot v_{t-1} + \alpha \cdot \nabla L(W)$$
$$W = W - v_t - \alpha \cdot \lambda \cdot W$$

Here v_t represents the velocity or momentum at time t, which is a combination of the previous velocity v_{t-1} and the current gradient descent ($\nabla L(W)$). β is the momentum parameter, controlling the influence of the previous velocity on the current update. λ is the weight decay factor, regulating the rate at which weights are decreased to prevent overfitting.

By incorporating momentum and weight decay, SGD with momentum achieves smoother and more stable weight updates, enhancing convergence speed and overall model performance.

Adaptive gradient Optimizer(Adagrad):

Adagrad is an optimizer that dynamically adjusts the learning rate for each weight based on its gradients. The learning rate for each weight is inversely proportional to the square root of the sum of its squared gradients up to the

current time step. Thus, larger sums of gradients result in smaller weight updates.

The weight update equation for Adagrad is as follows:

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}+\sigma}} \cdot \nabla L(w_i)$$

Where $w_{t+1,i}$ is the updated value of weight w_i at time step $t+1$, $w_{t,i}$ is the current value of weight w_i at time step t, η is the learning rate, $G_{t,ii}$ is the sum of squares of gradients with respect to w_i up to time step t, forming a diagonal matrix G_t, σ is a small constant (typically added for numerical stability).

In this equation, the learning rate is scaled by the square root of the sum of squared gradients. This scaling effectively reduces the learning rate for weights with large accumulated gradients, and increases it for those with smaller ones. This adaptive adjustment allows Adagrad to converge more quickly and effectively in many scenarios.

RMSProp gradient:

RMSprop employs a moving average of squared gradients to normalize gradients, ensuring balanced step sizes (momentum). This prevents exploding gradients for large gradients and alleviates vanishing gradients for small ones. The update equations for RMSprop are:

$$v_{t+1} = \beta \cdot v_t + (1-\beta) \cdot (\nabla L(w))^2$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_{t+1}+\epsilon}} \cdot \nabla L(w)$$

Where v_t is a moving average of squared gradients at time step t, β is a decay rate parameter, typically set close to 1, η is the learning rate, $\nabla L(w)$ is the gradient of the loss function with respect to the weights w and ϵ is a small constant (typically added for numerical stability).

In these equations, v_{t+1} is updated as a weighted sum of the previous moving average v_t and the squared gradient of the loss $(\nabla L(w))^2$. Then, the weights w are updated using the normalized gradient, where the learning rate η is scaled by the square root of the moving average of squared gradients plus a small constant η. This normalization ensures that the step size of the update is adapted according to the magnitude of the gradients, allowing for more stable and efficient optimization.

Adam Optimizer

The Adam optimizer combines both the first and second moments of loss derivatives to update the weights. The weight update equations are as follows:

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot (\nabla L(w_i))$$
$$v = \beta \cdot v + (1-\beta) \cdot (\nabla L(W))^2$$
$$w_t = w_{t-1} - \alpha \cdot \left(\frac{\sqrt{v}}{\eta}\right)$$

Where m_t is the first moment of the gradient at time step, β_1 is the decay rate for the first moment, typically set close to 1, $\nabla L(w_i)$ is the gradient of the loss function with respect to the weights w_i, v is the second moment (uncentered variance) of the gradient, β is the decay rate for the second moment, typically set close to 1, $\nabla L(W)$ is the gradient of the loss function with respect to all the weights W, W is the updated weight at time step t, α is the learning rate , $\sqrt{v}$ and η are used for normalization, typically with a small constant added for numerical stability.

In these equations, m_t and v are updated as exponentially decaying moving averages of the gradient and its squared value, respectively. Then, the weights w_t are updated using these moving averages and the learning rate α, with normalization to adapt the step size based on the magnitude of the gradients and the second moment. This integration of first and second moments enables Adam to efficiently optimize weights across various scales and dimensions, rendering it widely utilized in practice.

Schedulers:

Learning rate, a critical hyper-parameter in Neural Network training, is dynamically adjusted by schedulers during training iterations to enhance convergence speed and overall performance. Starting with a higher learning rate and gradually reducing it aids convergence acceleration. Common schedulers include:

1. One-cycle learning rate:

This scheduler gradually increases the learning rate in the initial training phase, then decreases it later. Concurrently, momentum parameters are adjusted, initially decreasing and later increasing.

2. Cosine Annealing:

Cosine Annealing starts with a high learning rate and gradually decreases it using a cosine function. This smooth adjustment facilitates better exploration of the optimization landscape and more efficient training.

S.No	Block Type	# Residual Blocks at layer	# Conv Channels at layer	Optimizer	Params	Final Acc
1	[BB, BB, BB, BB]	[2,3,2,2]	[64,128,192,256]	SGD+M	4.47M	**93.32**%*
2	[BB, BB, BB, BB]	[2,3,2,2]	[64,128,192,256]	Adagrad	4.47M	90.55%
3	[BB, BB, BB, BB]	[2,3,2,2]	[64,128,192,256]	RMSProp	4.47M	89.130%
4	[BB, BB, BB, BB]	[2,3,2,2]	[64,128,192,256]	Adam	4.47M	93.05 %
5	[BB, BB, BB, BB]	[2,2,2,2]	[64, 128, 232, 268]	Adam	4.99M	81.03%
6	[BB, BB, BB, BB]	[2,2,2,2]	[64, 128, 232, 268]	SGD+M	4.99M	**95.55**%
7	[BN, BN, BN, BN]	[2,2,2,2]	[64, 118, 178, 256]	SGD+M	4.99M	91.97%
8	[BN, BB, BB, BN]	[2,2,2,2]	[64, 128, 232, 256]	SGD+M	4.99M	92.39%
9	[BB, BN, BN, BB]	[2,2,2,2]	[64, 100, 160, 256]	SGD+M	4.99M	92.73%
10	[BN, BN, BN, BN]	[3,4,6,3]	[64, 96, 128, 190]	SGD+M	4.98M	**95.51**%
11	[BN, BN, BN, BN]	[3,4,6,3]	[64, 96, 128, 190]	Adam	4.98M	93.37%*
12	[BN, BN, BN, BN]	[3,8,36,3]	[64, 72, 74, 90]	SGD+M	4.99M	**93.82**%*

Table 1: Trained model(s) summary[* Trained for only 130 epochs][BB = Basic Block, BN = Bottle Neck]

Results

The experiment examined how altering convolutional channel quantities per layer affects model accuracy, employing cross-entropy loss with a maximum learning rate of 0.01 over 200 epochs.

Impact of More Convolutional Channels on Accuracy:

The results underscore the pivotal role of channel quantity in achieving high accuracy on the CIFAR-10 test dataset. Models with higher channel counts consistently demonstrated superior performance through meticulous evaluation.

For instance, a model with fewer channels exhibited an average test dataset accuracy of less than 90%. Yet, increasing channel size beyond 200 in the final layer notably enhanced accuracy, surpassing the 90% threshold.

Impact on accuracy with change in number of Residual Blocks in each layer:

The CIFAR-10 dataset, known for its low-resolution images, posed a unique challenge: deeper neural networks didn't always result in improved performance. Despite employing various ResNet architectures such as ResNet 18, 56, 101, and 156, increasing network depth didn't consistently boost accuracy. Surprisingly, ResNet models with depths ranging from 18 to 156 layers achieved comparable accuracies exceeding 95%. This unexpected finding suggests a nuanced relationship between network depth and CIFAR-10 performance.

To adhere to the specified 5 million parameter limit while maximizing accuracy, a strategic approach balanced channel numbers in each convolutional layer with network depth. Careful adjustment of these architectural parameters resulted in finely-tuned models, achieving high accuracy within resource constraints. This meticulous optimization process highlights the importance of thoughtful design decisions for optimal performance.

Residual Block Style:

Thorough evaluation of Residual Block styles — standard, Bottleneck, or a combination — revealed the standard Residual Block as the superior choice. Despite the potential effectiveness of Bottleneck Blocks in deeper networks, compatibility issues arose due to lower image resolution and parameter constraints. Combining both styles yielded similar results, with the standard block exhibiting more efficient parameter usage.

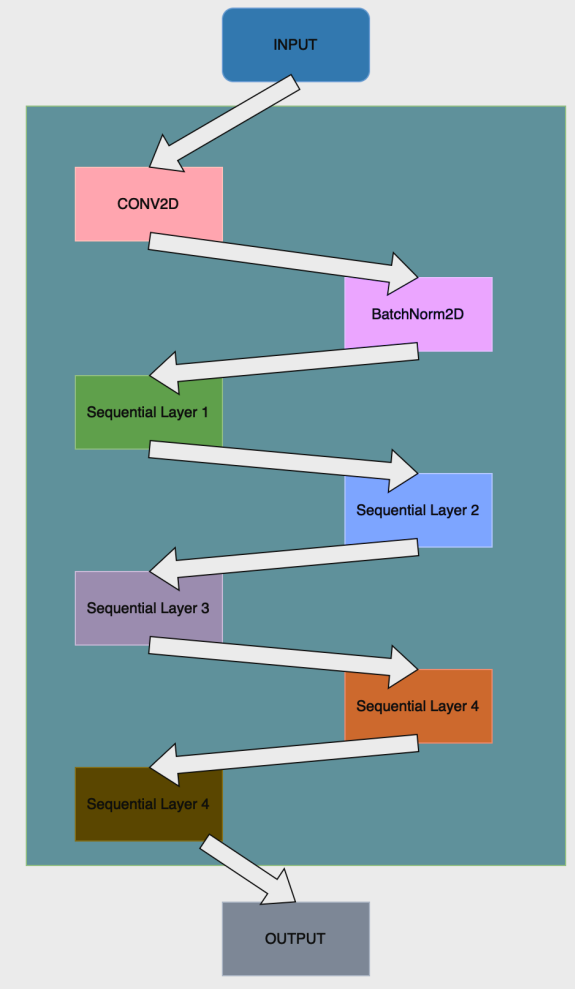
Impact of Different Optimizers:

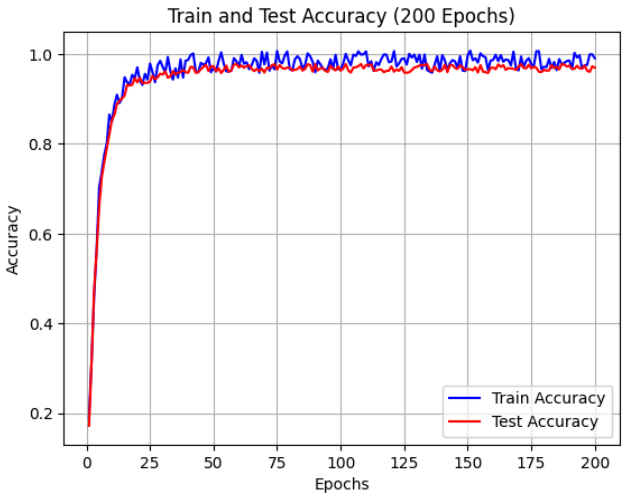
Optimizers are pivotal in minimizing network losses during training. SGD with momentum (SGD+M) emerged as the top performer, closely followed by Adam. SGD's effectiveness in classification problems likely contributes to its superior performance in this context.

Model Architecture:

Model 6, featuring four Residual layers with standard Residual blocks, achieved optimal performance. Initial Convolutional channels were set at 64, 128, 232, and 268. Employing a Cosine Annealing scheduler with SGD optimizer, momentum, and weight decay significantly enhanced classification accuracy.

While models 1, 10, and 12 showed potential to outperform Model 6, hardware limitations restricted their training duration to fewer than 35 epochs.





Conclusion

Aggressive adjustments to hyperparameters and optimizers enabled the development of a final model with fewer than 5 million parameters. Leveraging SGD with momentum (SGD+M), the model attained an impressive accuracy of 95.55% on the CIFAR-10 test dataset. Surprisingly, experimentation showed that increasing channel numbers in shallow networks and deepening the network with reduced channels produced comparable results. This flexibility highlights the significance of thoughtful parameter tuning and optimization strategies in model development.

References

[1] He, Kaiming and Zhang, Xiangyu and Ren, Shaoqing and Sun, Jian 2016. Deep Residual Learning for Image Recognition. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 770-778, doi: 10.1109/CVPR.2016.90.

[2] Srivastava, Rupesh Kumar and Greff, Klaus and Schmidhuber, Ju ̈rgen Highway Networks, 2015, https://doi.org/10.48550/arxiv.1505.00387

[3] Huang, Gao and Liu, Zhuang and Van Der Maaten, Laurens and Weinberger, Kilian Q. Densely Connected Convolutional Networks, 2017 IEEE Confer- ence on Computer Vision and Pattern Recognition (CVPR).

[4] Bishop, Christopher M and others Neural networks for pattern recognition, 1995, Oxford university Press.

[5] Github. 2017. Train CIFAR10 with PyTorch. https://github.com/kuangliu/pytorch-cifar.

[6] Stackoverflow. 2016. Is it a good learning rate for Adam method. https://stackoverflow.com/questions/42966393/is-it-good-learning -rate-for-adam-method

[7] PyTorch. 2023. Cosineannealinglr. https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CosineAnnealingLR.html

Acknowledgments

We express our gratitude to Professor Chinmay Hegde for his invaluable guidance, mentorship, and expertise throughout this project. We extend our thanks to the staff and facilities at New York University for providing essential resources for project completion. Additionally, we acknowledge the assistance of ChatGPT in generating content for specific sections of this report.