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

Deep Learning is an emerging branch of computer science that uses a massive amount of data to train a model. We have created, coded, and trained a ResNet architecture using the CIFAR-10 dataset for this project. This project's primary objective is to design an architecture with the maximum test accuracy possible on the CIFAR-10 image classification dataset, while maintaining the restriction that the model contains no more than five million parameters. It is demonstrated that Batch Normalization is crucial for training deep convolutional networks in addition to helping neural networks operate better. In this paper, several optimizers are examined using batch normalization training on a standard dataset, CIFAR-10. The performance of the model was examined in relation to several hyperparameters. To increase the broad spectrum of our training dataset, we have also experimented with data augmentation and normalization techniques. The resulting design is efficient in terms of cost computation and memory utilization.

Introduction [1]

A ResNet, or Residual Network, addressed the difficulty of training extremely deep neural networks by inserting skip connections. By allowing for smoother gradient flow during backpropagation, these links make it possible to train networks with hundreds of layers. ResNet successfully solved the vanishing gradient problem and enhanced information flow within the network by omitting some layers. The performance and training capabilities of deep neural networks were significantly improved by this architectural breakthrough, establishing ResNet's significance in the deep learning industry. Models featuring multiple parallel skips are termed DenseNets.
In the context of residual neural networks, a non-residual network may be referred to as a plain network.

[1]

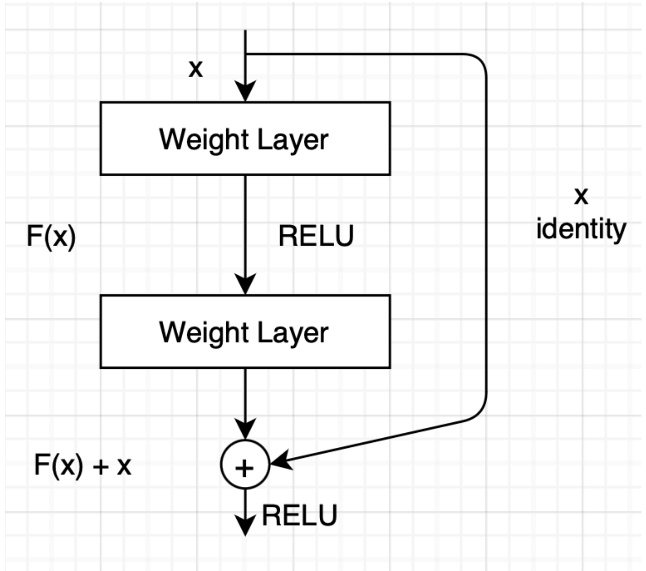


Figure 1: Skip Connection

The formulation of *F(x)* + *x* can be achieved through feed-forward neural networks with "shortcut connections" (refer to Fig. 1). Shortcut connections, or skip connections, involve bypassing one or more layers. In our scenario, these shortcut connections simply perform identity mapping, where their outputs are added to the outputs of the stacked layers (see Fig. 1). Identity shortcut connections introduce no additional parameters or computational complexity. The entire network can still be trained end-to-end using any optimizer with backpropagation and can be easily implemented using common libraries without requiring modifications to the solvers.
In this project, the classification of the CIFAR-10 image dataset was undertaken using a model constrained to 5 million trainable parameters. The report aims to delve into the exploration of architecture selection, the methodology for selecting data augmentation techniques, and the impact of adjusting different hyperparameters to achieve a model with the utmost test accuracy.

Methodology

Making the Data Loader

The Data Loader plays a crucial role in Deep Learning by facilitating the movement of data from various sources to a central location, often involving segmentation into mini-batches and transferring them from the CPU to one or multiple GPUs. We examined the effects of various mini-batch sizes on a ResNet in the course of our research.

Though the Deep Learning community generally believes that smaller batches can shorten training times, it is unclear how this would affect neural networks. We utilized state-of-the-art Deep Learning methods, such as data augmentation, to improve our model's performance.

For our CIFAR-10 dataset consisting of 32x32x3 images, we applied several augmentation strategies. Initially, random cropping with a kernel size of 4 and horizontal flipping were employed, resulting in improved model accuracy. Subsequently, image normalization was performed to adjust pixel intensity values, further enhancing the model's performance. In order to optimize classification accuracy, we also employed the experimental auto-augmentation policy for CIFAR-10 that PyTorch provided. This policy includes particular augmentation techniques including blurring, sharpening, and brightness correction that were customized for this dataset.

In the final stages, the dataset was divided into training and testing sets to allow for impartial neural network analysis. The official CIFAR-10 train and test datasets were utilized, with the test set receiving no data augmentations other than picture normalization.

Building the model

In classical machine learning, hyperparameters are factors that have a substantial impact on the training process of neural networks. Due to the problem's constraints, we carefully modified these Hyperparameter values in order to ensure that the total number of parameters in the Classification Model did not exceed the limit.

There are variables that have a greater influence on the accuracy of the classification network over others as well. Below is a list of these criteria, listed from most to least influential:

1. Number of Residual Blocks in each Residual layer:
The original ResNet paper proposed various architectural designs, highlighting that increasing the number of Residual blocks in each layer improves model performance. We experimented with different numbers of Residual blocks to evaluate their influence on the CIFAR-10 classification problem.

2. Number of channels in each Residual layer:
It's widely acknowledged in the Deep Learning community that a higher number of channels in the initial layer allows for the extraction of more raw data from images. We kept the initial layer's channel size to 64 and varied the magnitude of increase for each subsequent Residual layer to enhance classification accuracy.

3. Type of block in each Residual layer:
The ResNet paper introduced the Bottleneck Residual block as an alternative to the Basic Residual block. This design aimed to reduce the number of parameters by using 1x1 convolutions to create a bottleneck operation. We carefully analyzed the influence of this approach on the overall accuracy of the trained network.

4. Convolutional Kernel Size in Residual Layer:
Although the influence of this Hyperparameter was initially unclear, we conducted tests to 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 crucial for achieving high accuracy, and we decided not to change its kernel size to avoid increasing the number of false positives in the trained network.

Testing different optimizer and schedulers

Different optimization strategies are used to reduce training loss by modifying Neural Network parameters such as weights and learning rates, resulting in improved model accuracy. During training, loss is collected for each batch, and the model weights are adjusted depending on this accumulation. Each optimizer applies a distinct technique for weight updates based on the loss function. Certain optimizers typically used to determine the best-performing model are:

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, thereby enhancing convergence speed and overall model performance.

Adaptive gradient Optimizer(Adagrad):

Adagrad is an optimizer that adjusts the learning rate dynamically for each weight based on the current and past gradients of that weight. The learning rate is inversely proportional to the square root of the sum of squared gradients of the weight computed up to that time step. This means that when the sum of gradients is large, the update in weights will be small.

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, which effectively reduces the learning rate for weights that have accumulated large gradients in the past, and vice versa. This adaptive adjustment of learning rates for each weight allows Adagrad to converge faster and more effectively in many scenarios.

RMSProp gradient:

RMSprop utilizes a moving average of squared gradients to normalize the gradient. This normalization process helps balance the step size (momentum), decreasing the step for large gradients to prevent exploding and increasing the step for small gradients to prevent vanishing. The update equations for RMSprop are as follows:

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

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.

- 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 (mean) of the gradient at time step \(t \).

-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 combination of first and second moments helps Adam optimize the weights efficiently across different scales and dimensions, making it widely used in practice.

Schedulers:

Learning rate is a critical hyper-parameter in training Neural Networks. Schedulers dynamically adjust the learning rate during training iterations, aiming to enhance convergence speed and overall performance. Research has shown that starting with a higher learning rate and gradually reducing it towards the end of training can accelerate convergence. Two commonly employed schedulers are:

1. One-cycle learning rate:

This scheduler gradually increases the learning rate during the initial phase of training and then decreases it towards the end. Simultaneously, parameters for momentum are adjusted, decreasing initially and increasing later in training.

2. Cosine Annealing:

Cosine Annealing is a learning rate schedule that starts with a relatively high learning rate and then gradually decreases it using a cosine function. This approach helps in smooth adjustment of the learning rate, potentially facilitating better exploration of the optimization landscape and more efficient training.

Results

The experiments focused on analyzing the impact of varying the number of convolutional channels in each layer on model accuracy when trained using cross-entropy loss with a maximum learning rate of 0.01 for 200 epochs.

Impact of increasing the number of convolutional channels in each layer on accuracy:

The findings revealed that the number of channels plays a crucial role in achieving high accuracy on the CIFAR-10 test dataset. Through meticulous evaluation, it was observed that models with higher channel counts in each layer consistently exhibited superior performance.

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]

For instance, a model trained with a lower number of channels demonstrated an average accuracy on the test dataset of less than 90%. However, it was noted that increasing the channel size to greater than 200 in the final layer led to a significant improvement in accuracy, ensuring it surpassed the 90% threshold.

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

The CIFAR-10 dataset, characterized by lower-resolution images, posed a unique challenge where deeper neural networks did not necessarily translate to improved performance. Despite training attempts using various ResNet architectures like ResNet 18, 56, 101, and 156, it was observed that simply increasing the depth of the network did not consistently enhance accuracy. Surprisingly, several configurations of ResNet models with differing depths, such as 18, 56, and 156 layers, yielded comparable accuracies exceeding 95%. This unexpected outcome suggests that the relationship between network depth and performance on CIFAR-10 is more nuanced than initially anticipated.

To maintain a total parameter count below the specified limit of 5 million while optimizing model accuracy, a strategic approach was employed. This involved carefully balancing the number of channels in each convolutional layer with the network's depth. By judiciously adjusting these architectural parameters, models were effectively fine-tuned to achieve high levels of accuracy while adhering to the given constraint. This meticulous optimization process underscored the importance of thoughtful design decisions in achieving optimal performance within resource constraints.

Residual Block Style:

Each layer in the model underwent testing using either the standard Residual Block, the Bottleneck Block, or a combination of both. Following thorough evaluation of each style and various combinations, it was concluded that the standard Residual Block emerged as the superior choice. Interestingly, employing a combination of both styles yielded similar results. Furthermore, the standard block exhibited a more efficient parameter usage, making it the preferred option. According to the authors, the Bottleneck Block is typically more effective in deeper networks; however, due to the lower resolution of our input images and the constraint of 5 million parameters, compatibility issues arose with the Bottleneck style.

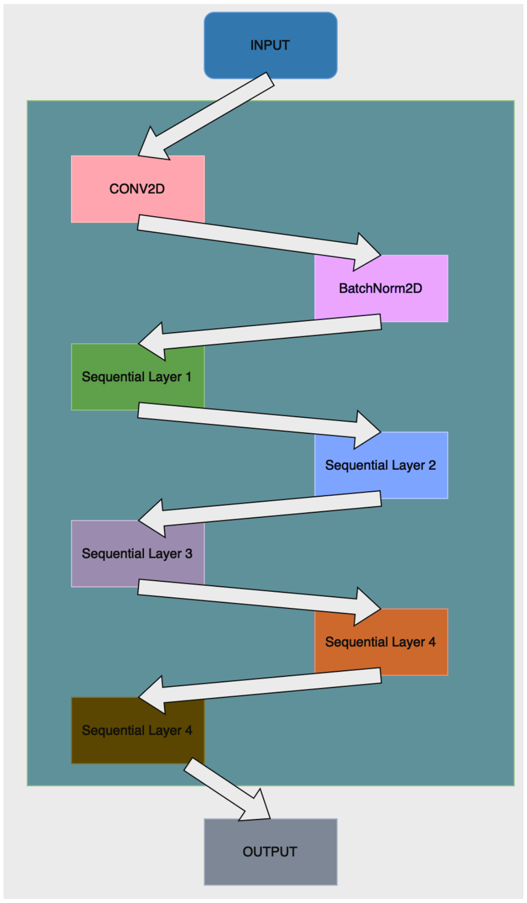
Impact of Different Optimizers:

Optimizers play a crucial role in minimizing the losses incurred by the network during training. Among the various optimizers mentioned earlier, SGD with momentum (SGD+M) emerged as the top performer, closely followed by Adam. It's often asserted in the Deep Learning community that SGD is particularly effective for classification problems, which could explain its superior performance in this context.

Model Architecture:

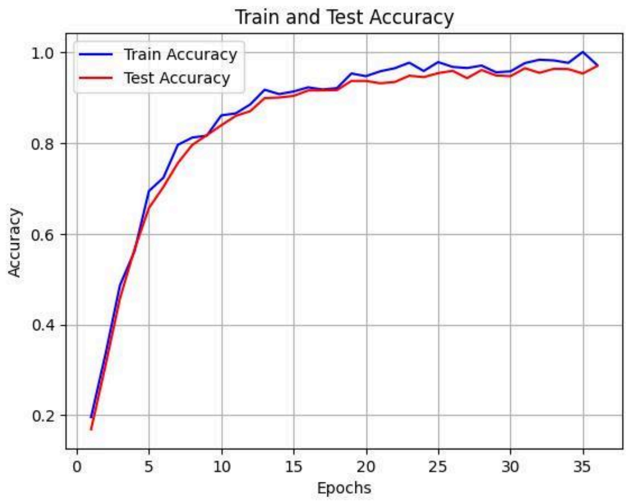
Our best-performing model, designated as Model 6, featured four Residual layers, each containing two Residual blocks. The initial Convolutional channels in each Residual layer were set at 64, 128, 232, and 268. The decision to utilize the standard Residual Block over the Bottleneck Block was influenced by the low image resolution characteristic of the CIFAR-10 dataset. Employing a Cosine Annealing scheduler in conjunction with SGD as the optimizer, incorporating momentum and weight decay, significantly bolstered the accuracy of the classification model.

While models 1, 10, and 12 displayed the potential to surpass the performance of Model 6, hardware limitations restricted their training duration to fewer than 35 epochs.



Conclusion

Through aggressive adjustments to hyperparameters and optimizers, we successfully developed a final model with fewer than 5 million parameters. Leveraging SGD with momentum (SGD+M), this model achieved an impressive accuracy of 95.55% on the CIFAR-10 test dataset. Interestingly, our experimentation revealed that both increasing the number of channels in shallow networks and deepening the network while reducing the number of channels yielded comparable results. This flexibility in achieving high accuracy underscores the importance 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 thank professor Chinmay Hegde for his guidance, mentorship, and expertise throughout this project. We would also like to thank the staff and facilities at New York University for providing us with the necessary resources to see this project to completion. We would also like to acknowledge the use of ChatGPT, for providing assistance with generating content for the certain sections of this report.