Deep Learning - Mini Project ResNet

Pratham Mehta (pm3483@nyu.edu), Aniket Sinha (aas10120@nyu.edu), Priyanshi Singh(ps4609@nyu.edu)
Department of Electrical and Computer Engineering, New York University

Our CodeBase: Link

Abstract

Recently, image classification using deep learning has gained immense popularity due to the high accuracy of its neural network models which require a lot of computational power. In this research, we carried out experiments on the CIFAR-10 image data set using a custom ResNet architecture with a limit of 5 million model parameters. To enhance the accuracy of our model with this limited parameter count, we utilized techniques such as hyper-parameter tuning, optimizer experimentation, data augmentation, and knowledge distillation.

Overview

Although training deep neural networks can be a daunting task, they demonstrate impressive results when it comes to image classification tasks. When compared to other similar neural networks, deep residual networks (ResNets) have the potential to improve accuracy and accelerate the training process. To attain high test accuracy on the CIFAR-10 image classification dataset without exceeding 5 million parameters, we devised a customized ResNet architecture. ResNets consist of N Residual layers, which consist of one or more residual blocks. Each residual block is made up of two convolutional layers that are connected by a skip link. The skip connection, illustrated in Figure 1, enables the model to skip some layers.

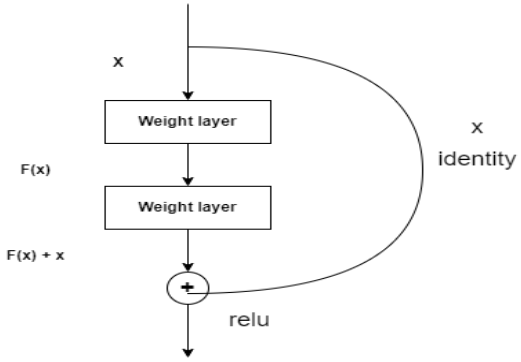


Figure 1: Residual Block
Source: Deep Residual Learning for Image Recognition [1]

The model's output with skip connection is H(x) = f(x) + x. Achieving high accuracy for our customized residual network requires a combination of architecture, training methods, and scaling strategy. To accomplish this, we employed three common techniques with parameter constraints. Firstly, we fine-tuned hyperparameters to adjust the number of ResNet layers. Secondly, we utilized data augmentation and increased the amount of data for model training. Lastly, we trained our model for a longer duration by increasing the number of epochs.

Resnet Enhances CNN Performance

The ResNet model was developed to address the issue of performance degradation that occurs in convolutional neural networks after a certain depth. This problem was solved by introducing residual blocks, which are the building blocks of the ResNet model. Each residual block comprises two convolution layers and a skip connection from the input to the output of the block. The ResNet model is made up of several layers, each of which consists of multiple residual blocks. As ResNets become deeper, the number of operations within a block is increased. An operation involves applying a convolution with batch normalization and ReLU activation to an input, except for the last operation in a block, which does not have a ReLU. ResNets are capable of addressing the issue of the vanishing gradient problem, among other problems. Sigmoid function causes vanishing gradients hindering weight updates and learning. ResNets solve this by enabling gradient flow through skip connections from later to earlier layers. Our unique architecture, depicted in the accompanying figure, incorporates this feature.

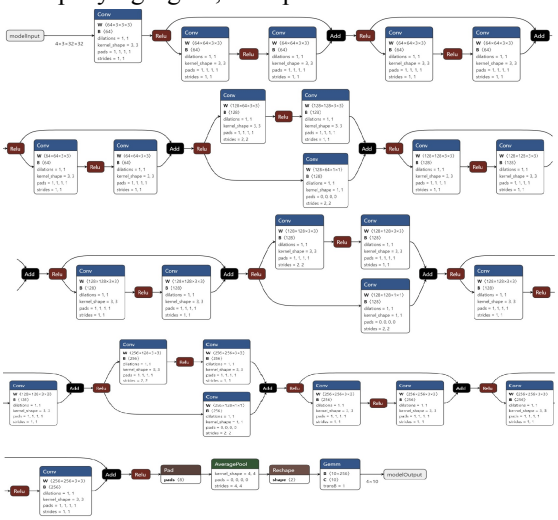


Fig: Architecture Diagram[12]

Summary

To achieve the highest accuracy with the Resnet model, we adjust several hyper-parameters, namely:
1. the number of layers (N),
2. the number of residual blocks in each Residual layer (Bi),
3. the number of channels in each Residual layer (Ci),
4. the convolutional kernel size in each Residual layer (Fi),
5. the skip connection kernel size in Residual layer I (Ki),
6. the average pool kernel size (P).

Our approach involved creating a model with four Convolution layers that was trained on the CIFAR-10 dataset for image classification. Our most successful model had an accuracy of 91.47%, utilizing 4,935,242 parameters. The hyperparameter values used in the model which had the highest accuracy are:

Learning Rate	0.01
Weight Decay	0.0001
Optimizer	Adadelta
Scheduler	SGD with Warm Restarts
N (Number of Layers)	4
B (Number of Blocks)	3, 3, 2, 3
C (number of Channels)	64, 128, 128, 256
K (Skip Kernet)	1, 1, 1, 1
F (Cov Kernel)	3, 3, 3, 3
P (Pool Kernet)	4
Number of Parameters	4,935,242 (less than 5 mill)

Methodology

The process involved in this study began by importing the CIFAR-10 dataset and dividing it into three groups: training, testing, and validation. To enhance the training set, we implemented two techniques to augment the data, namely horizontal flipping and random cropping. We then experimented with multiple ResNet hyperparameters, kernel sizes, optimization methods, and epoch numbers to determine the most effective approach.

ResNet Hyper-parameters

The hyperparameters of ResNet, namely K (skip kernel), F (Conv kernel), and P (pool kernel), were chosen based on the values used in the Wide Residual Networks paper. In terms of the number of layers (N) and residual blocks (B), we initially tested a 3-layer ResNet model with 1, 2, and 1 configuration of residual blocks. Subsequently, we experimented with different numbers of residual blocks and analyzed the outcomes. Our findings indicate that the best results were achieved with a four-layer network.

Data Augmentation

Data augmentation is a process that improves the generalization of a model by creating variations of the original image data. One technique of data augmentation is Random Cropping, where a random subset of an image is generated to enable the model to learn objects of interest that may not be wholly visible or at the same scale in the training data. This method also allows for the analysis of an image in different areas using differently sized windows. Another technique is Random Horizontal Flipping, which mirrors the image horizontally with a probability p. The rationale behind this is that an object should be recognizable whether in its original or mirrored form. These techniques help to generate more diverse data for training and improve the performance of machine learning models.

Data Transformation

Data transformation involves two steps.

- Firstly, conversion to a PyTorch tensor takes place, which scales the image by a factor of 255.
- Secondly, normalization is applied to adjust the image values to have a mean of 0.0 and a standard deviation of 1.0. This is achieved by subtracting the mean RGB values of the dataset (0.4914 for Red, 0.4822 for Green, and 0.4465 for Blue) from their respective color channels and dividing by their standard deviation values of the RGB Pixels (0.2023, 0.1994, and 0.2010). This normalization process helps to prevent any individual feature from dominating the training process and ensures faster convergence during model training.

Selecting Optimizers and Learning Rate

The selection of optimizers and learning rate plays a crucial role in deep learning models. Optimizers are algorithms that aim to minimize errors when mapping inputs to outputs, leading to improved accuracy and reduced overall loss. There are numerous well-known optimizers to choose from.

1. The Stochastic Gradient Descent (SGD) algorithm involves selecting random batches of data instead of using the entire dataset for each iteration. The data is shuffled randomly at each iteration to approach an approximate minimum based on the momentum w and learning rate parameters. This method helps to avoid overfitting and improve efficiency in training models.

2. Adaptive Gradient Descent, or Adagrad, adjusts the learning rate for each iteration based on the changes in parameters during training. When there are more changes in parameters, the learning rate is adjusted more minimally. Adagrad eliminates the need for manually adjusting the learning rate and can converge at a faster rate than other methods.

3. AdaDelta is an optimizer that improves upon the AdaGrad and RMSprop optimizers by using adaptive learning techniques. Its purpose is to overcome the limitations of these two optimizers, which require manual setup of the learning rate and suffer from a decreasing learning rate over time. AdaDelta addresses these issues by maintaining two state variables, which are the leaky averages of the second moment gradient and the second moment of change in parameters within the model.

4. Adam is an optimization algorithm that is based on Stochastic Gradient Descent. It differs from Stochastic Gradient Descent in that it modifies the learning rate for each weight in the network separately. Adam combines features from Adagrad and RMSprop optimizers, utilizing the second moment of the gradients to compute the mean of the uncentered variance. This allows for quicker computation time compared to traditional Stochastic Gradient Descent, which prioritizes individual data points.

5. Nesterov Accelerated Gradient (NAG) is a variant of SGD that uses a "look ahead" approach to update the gradient. It updates the gradient based on the predicted future position of the parameters, which allows it to converge faster than traditional SGD.

6. RMSprop is an optimization algorithm used in deep neural networks. It adapts the learning rate for each parameter based on the magnitude of recent gradients. It reduces oscillations and allows for a smoother and faster convergence during the optimization process.

We experimented with three different optimizers, namely AdaDelta, AdaGrad, and Adam, and tested our models using these optimizers. The graph shows the accuracies achieved by our models, which were the best accuracies for each optimizer with different learning rates. We started training with a learning rate that decreased as the number of epochs increased, and continued for 300 epochs with a learning rate of 0.01 and weight decay of 0.0001. This technique is known as a Learning Rate Scheduler, which helps in achieving faster convergence with higher accuracy. There are two types of Learning Rate Schedules, Step-wise Decay, and Reduce on Loss Plateau Decay. For our model, we used the Stepwise Decay approach..

In our approach, we have implemented the Cosine Annealing technique, which is a method for adjusting the learning rate schedule during training. This technique involves starting with a high learning rate that decreases quickly to a minimum value and then increases rapidly again. This resetting of the learning rate simulates the restart of the learning process, and the use of previously successful weights as the starting point is called a "warm restart." In contrast, a "cold restart" involves starting with a new set of small random numbers. We also experimented further to see how our model gets trained in 300 epochs by changing Learning Rate to 0.001.

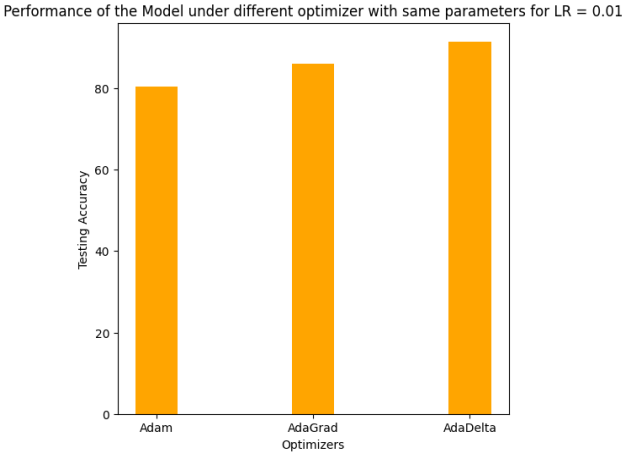
Result

The objective of this research is to classify images in the CIFAR-10 dataset by experimenting with various hyper-parameters of ResNet architecture and implementing

data augmentation techniques. Our study resulted in the highest accuracy of 91.61% by training the dataset for 300 epochs with adadelta optimizer and a learning rate of 0.01. Our model comprises four convolutional layers, each having a different number of residual blocks. However, since we only examined three optimizers, it could happen that utilizing advanced optimizers and data augmentation methods could result in even higher accuracy.

In our experiments, we also found that the effect of changing the learning rate from 0.01 to 0.001 varied the model accuracy depending on the optimization algorithm used. Specifically, for the Adam and AdaGrad algorithms, reducing the learning rate from 0.01 to 0.001 led to better accuracy as it allowed the model to make smaller and more precise updates to the parameters during training. However, for the AdaDelta algorithm, reducing the learning rate further from an already optimal(0.01) or too low original learning rate led to slower convergence or even worse performance. Since, AdaDelta optimization had the highest accuracy, we ran our model again with 0.1 Learning Rate as well

Learning Rate	Adam	Adagrad	AdaDelta
LR: 0.01	80.46	86.07	**91.47**
LR: 0.001	89.19	88.37	75.72



.

Learning Rate	0.1	**0.01**	0.001
Test Accuracy	91.03	**91.47**	75.72

To summarize, the optimal learning rate for a given optimization algorithm depends on several factors such as the complexity of the model, the size of the dataset, and the nature of the task. It is a good practice to experiment with different learning rates and monitor the training process to find the optimal value that maximizes the accuracy on a validation set. Moreover, it is important to take into account the characteristics of the optimization algorithm being used to determine the appropriate learning rate to achieve the best performance.

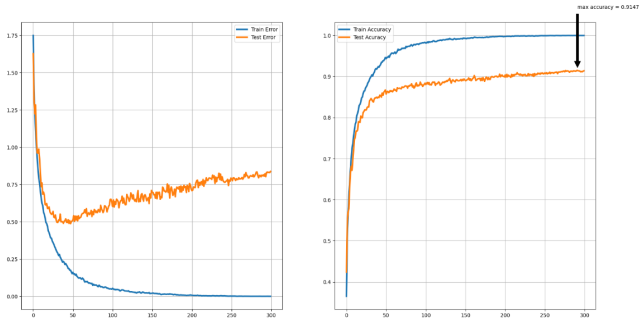


Fig: 1.train loss vs test loss 2. train accuracy vs test accuracy

References

[1] He, Kaiming et al. "Deep Residual Learning for Image Recognition." CoRR, abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385

[2] Liu, K. (2018). pytorch-cifar. GitHub repository. Retrieved April 14, 2023, from https://github.com/kuangliu/pytorch-cifar

[3] PyTorch. (2022). PyTorch 1.9.0 documentation. Retrieved April 14, 2023, from https://pytorch.org/docs/stable/index.html

[4] Bansal, S. (2019). ResNets for CIFAR-10. Towards Data Science. Retrieved from https://towardsdatascience.com/resnets-for-cifar-10-e63e900524e

[6] Neptune. (2022). Data Augmentation in Python. URL: https://neptune.ai/blog/data-augmentation-in-python

[7] GeeksforGeeks. (n.d.). Residual Networks (ResNet) - Deep Learning. Retrieved from geeksforgeeks.org/residual-networks-resnet-deep-learning/

[8] Krizhevsky, A. (2009). CIFAR-10 Dataset. Retrieved from https://www.cs.toronto.edu/kriz/cifar.html

[9] Papers With Code. (n.d.). Cosine Annealing. Retrieved from https://paperswithcode.com/method/cosine-annealing

[10] Learning Rate Schedules and Adaptive Learning. URL:https://towardsdatascience.com/learning-rateschedules-and-adaptive-learning-rate-methods-fordeep-learning-2c8f433990d

[11] Brownlee, J. (2019). Understand the Dynamics of Learning Rate on Deep Learning Neural Networks. Retrieved from https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[12] Architecture Diagram was drawn using Netron API. Retrieved from https://netron.app/