CIFAR-10 Classification using ResNet18
DL Mini Project

Link to GitHub
by
Ashwath Shankarnarayan, Suriya Prakash Jambunathan, Shubham Halyal

Abstract

The aim of this project is to optimize results using the ResNet architecture and a comprehensive hyper-parameter tuning method. The architecture is strictly limited to 5 million parameters, without pre-trained weights, and only using the CIFAR-10 dataset. The goal is to achieve high efficiency and effectiveness without compromising computational efficiency. The focus on hyper-parameter tuning allows for optimal parameter settings. The project aims to develop a cutting-edge deep learning model that delivers exceptional performance while adhering to strict computational constraints. By leveraging ResNet architecture and applying thorough hyper-parameter tuning, the model can offer unparalleled accuracy and efficiency, making it a valuable asset for various applications.

Overview

It is widely acknowledged by researchers that increasing the depth of convolutional neural networks can lead to improved performance due to the larger parameter space available for exploration. However, it has been observed that there is a point of diminishing returns, beyond which the generalization capability of the network can actually begin to degrade.

This limitation was encountered by the VGG architecture, which was unable to achieve the desired depth due to diminishing performance. To address this issue, Residual Networks (ResNets) were introduced, which incorporate skip connections to allow for the direct flow of gradients between later layers and initial filters. This helps to mitigate the problem of vanishing gradients, which occurs when the gradients propagated back through the network become very small, making it difficult to update the weights and impeding the learning process.

Data Analysis

The dataset distribution is a critical topic when it comes to machine learning. An imbalanced dataset (different number of examples for each class), may hamper our model from achieving good overall accuracy. Each class should contain the same number of examples for the dataset to be perfectly balanced.

The CIFAR10 dataset is composed of 60000 32x32 color images (RGB), divided into 10 classes. We are splitting 50000 train images as 40000 training set and 10000 as validation set. 10000 images for the test set.

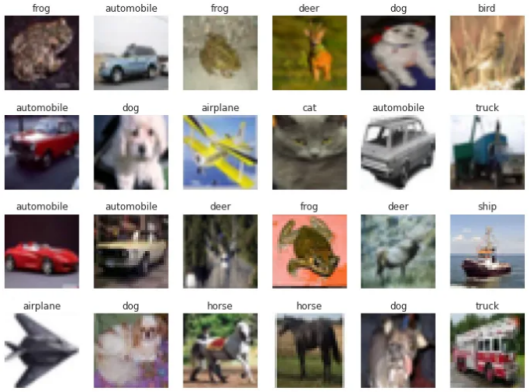

Figure 1: Random Images from CIFAR-10

.

Data Augmentation

Deep neural networks are known to be susceptible to overfitting, which makes it important to have sufficient and varied training data. One approach to addressing this is through data augmentation, which involves creating new variations of the existing data in the training set. This can help to increase the effective size of the training set and introduce errors that serve as a form of regularization.

CIFAR-10 dataset offers only 50000 training images distributed in 10 classes. For each class we have very less data points. Data augmentation often involves applying transformations such as rotating or cropping. For this project, we are using Random Cropping with size: 32 & padding: 4 and Random Resized Cropping with size: 32 & scale: (0.8, 1.0), ratio: (0.8, 1.2). One common transformation is horizontal flipping, which we are applying with the factor p: 0.5. This involves creating a mirror image of the original by flipping it horizontally.

For the data normalization we selected parameters as follows:
mean: (0.4914, 0.4822, 0.4465)
std: (0.2023, 0.1994, 0.2010)

Architecture

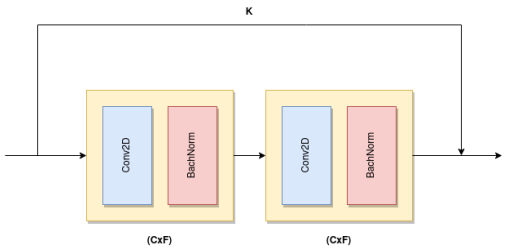

Figure 2: Basic block representation

.

The key component in ResNet models is a residual block that implements:

$$ReLU(S(x) + F(x)) \tag{1}$$

where S(x) refers to the skipped connection and F(x) is a block that implements $conv- > BN- > relu- > conv- > BN$

here, "BN" stands for batch normalization.

Chaining such blocks serially gives a deep ResNet. Hyperparameters (design variables) in such architectures include:

- C_i , the number of channels in the i^{th} layer.
- F_i , the filter size in the i^{th} layer
- K_i , the kernel size in the i^{th} skip connection
- P , the pool size in the average pool layer,

The vanilla version of ResNet-18 uses a little over 11 million parameters. The original ResNet-18 had the block repetitions [2, 2, 2, 2], we made it [1, 1, 1, 1] to fit into the 5 million threshold. Even though this is a version of a well-known model, due to it having very less number of repetitions, it resulted in a very shallow network.

For this project we are using modified ResNet18 architecture. We are using total 31 layers of Conv2D, 31 layers of BatchNorm, 12 layers of BasicBlock (skipped connection), and 1 Linear layer at the end of model. Following the contraints on the number of parameters we are getting total 4,891,338 trainable parameters.

Methodology

Architecture Selection

We pursued a novel approach to address the challenge of optimizing the ResNet-18 architecture while adhering to the stringent 5 million parameter limit. Our strategy involved increasing the channel sizes while maintaining a deeper neural network, resulting in a new model which we named as the Deep Narrow version of ResNet-18. This model comprised 50 additional convolutional layers and featured a channel size progression of 32, 64, 128, and 256, with block repetitions of [13, 9, 3, 3].

Furthermore, we devised another architecture, which we named it the ZigZag version of ResNet-18, which incorporated a more diversified channel size progression. This approach involved starting with 64 channels, increasing to 128, then to 256, followed by a drop to 128, 64, before a bounce-back to 128, with a final stop at 256. This configuration introduced more variety into the model, utilizing seven distinct block structures, in contrast to the four unique block structures in the original ResNet-18 and Deep Narrow version. Our results indicated that the ZigZag model outperformed the other architectures in terms of accuracy.

Hyperparameter Selection

The hyper-parameter tuning process of our deep learning model involved an extensive search for optimal values of batch size, optimizer, and data augmentation techniques. We experimented using WandB(to keep track of your hyperparameters, system metrics, and predictions) by performing different sweeps with different models with various batch sizes ranging from 32 to 128, multiple optimizers including SGD, SGD with Nesterov, Adam, Adamax, Adagrad, RMSProp, and Adadelta, as well as different data augmentation techniques. From the results obtained, we identified the top-3 best-performing augmentation techniques and moved them to the next stage of tuning. After careful analysis, we determined that the best batch size was 32 and the optimal optimizer was SGD with Nesterov.

We continued the tuning process by exploring several learning rate values and ran our model for 100 epochs using the best learning rate. During this process, we discovered that fixing the learning rate throughout the training process led to suboptimal results in the later epochs. Thus, we experimented with several Learning Rate Schedulers, including CyclicLR and ReduceLROnPlateau, and observed that the latter scheduler provided the best results. We further refined the hyperparameters of the ReduceLROnPlateau scheduler, including the factor, patience, and mode, and arrived at optimal values of 0.1 factor, 5 patience, and 0.01 SGD Learning Rate.

We also conducted experiments to fine-tune the SGD Optimizer to achieve its best performance by searching for optimal values. We determined that a momentum of 0.8 and a weight decay of 0.0005 provided the best results. We conducted all these experiments for both the ZigZag and Deep Narrow versions of ResNet, and we observed that the ZigZag ResNet provided the best overall performance. Ultimately, we achieved a top accuracy of 93.5% by optimizing the model's hyperparameters to minimize loss and maximize accuracy.

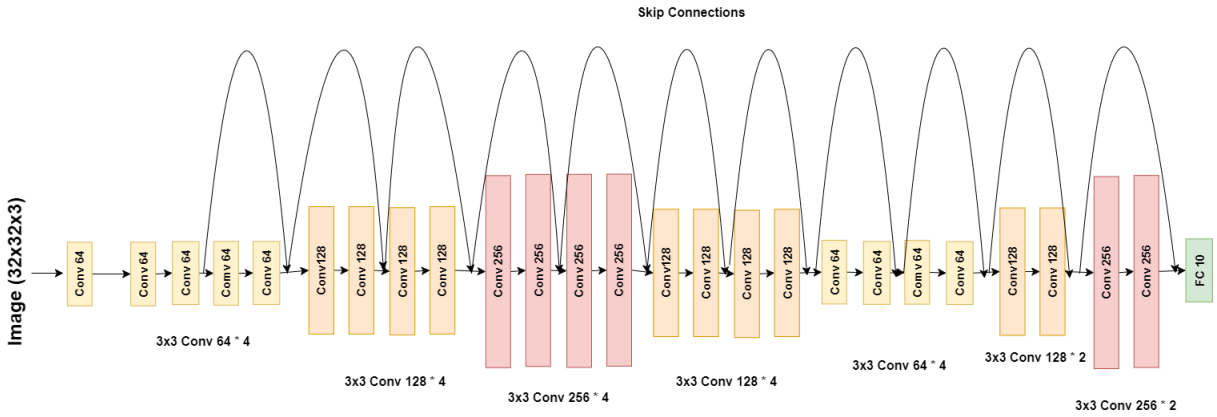

Figure 3: Zig-Zag ResNet Model.

	batch_size	augmentation	optimizer	train_loss	valid_loss	train_acc	valid_acc
0	32	no	SGD	0.856368	0.552547	85.174	81.19
1	32	no	SGD with Nesterov	0.824306	0.551144	85.348	81.77
2	32	no	ASGD	1.049168	0.810727	81.768	73.52
3	32	no	Adam	2.585961	1.659058	54.276	40.51
4	32	no	Adamax	1.840848	1.019998	67.942	64.22
...	...	...	...	...	...	...	...
121	128	random_erasing	SGD with Nesterov	1.218360	1.266651	78.466	63.24
122	128	random_erasing	ASGD	2.049126	1.058681	63.284	62.90
123	128	random_erasing	Adam	1.753814	1.043496	69.298	64.62
124	128	random_erasing	Adamax	1.449330	0.801869	74.776	72.80
125	128	random_erasing	Adadelta	2.315948	1.161275	58.940	57.78

Figure 4: Different optimizers and augmentations.

	batch_size	augmentation	optimizer	train_loss	valid_loss	train_acc	valid_acc
1	32	no	SGD with Nesterov	0.824306	0.551144	85.348	81.77
7	32	random_crop	SGD with Nesterov	1.147717	0.578310	80.240	80.63
13	32	random_horizontal_flip	SGD with Nesterov	0.946680	0.597674	83.478	79.60
19	32	random_rotate	SGD with Nesterov	1.161062	0.609516	79.840	78.65
25	32	random_resized_crop	SGD with Nesterov	1.084598	0.552147	81.040	80.83
31	32	random_affine	SGD with Nesterov	1.375324	0.630698	76.018	78.55
37	32	random_erasing	SGD with Nesterov	1.077594	0.710151	81.130	76.79

Figure 5: Different augmentation techniques.

Additional Experiments

After achieving an accuracy of 93.5% with the ZigZag ResNet, the team further explored the logs of the model training to improve the model's performance. They observed that the learning rate became too small after a certain point in epochs, and the scheduler only decreased the learning rate when the model was performing worse. To address these issues, they created a new scheduler called ZigZagLROnPlateau that controlled learning rate changes in both directions of model performance change. This new scheduler helped the team achieve good accuracies in the first few epochs, but the improvement after that point saturated as the learning rate became too small. The team then explored the Restarts and WarmRestarts version of several different schedulers, but faced overfitting issues, as the training accuracy was upwards of 99-99.5%, while the validation/test accuracies were capped at 92.94%.

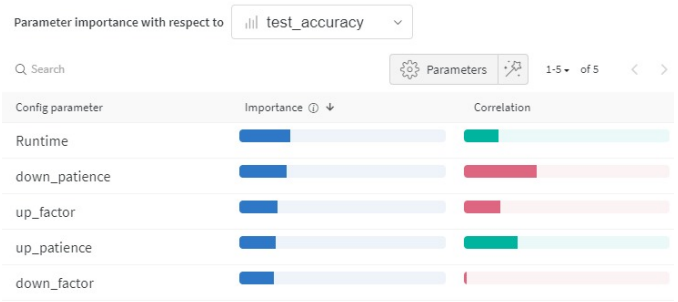

Figure 6: Parameter importance for patience and factor.

To address the issue of overfitting, the team decided to use a metric that maximized accuracy and minimized loss. However, they still faced overfitting issues, as maximizing accuracy alone led to model overfitting. To further modify the metric, they penalized the improvement in train accuracy and supported the increase in train loss by a small factor. This approach, while not very recommended, helped the team decrease overfitting a bit, and they were able to achieve close to the previous best result, with around 92% accuracy, but the train accuracy was down to 96-98%. Overall, the team's careful exploration and experimentation with various hyperparameters, optimizers, schedulers, and metrics helped them achieve a high accuracy of 93.5%, and also helped them mitigate overfitting issues.

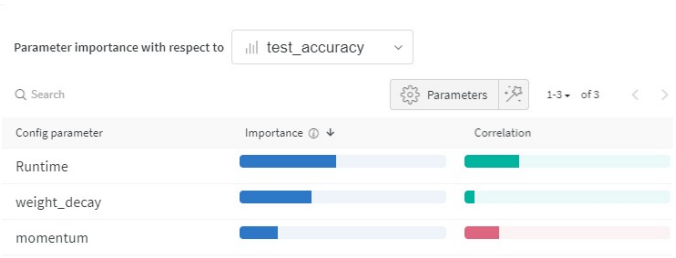

Figure 7: Parameter importance for weight decay and momentum.

Results

After conducting a series of experiments, we have identified that the most optimal model for our task is the ZigZag ResNet architecture. The model is trained using a batch size of 32 and a combination of augmentations including RandomCrop with a size of 32 and padding of 4, RandomHorizontalFlip with a probability of 0.5, and RandomResizedCrop with a size range of 32 and scale ratio of 0.8-1.0 and an aspect ratio range of 0.8-1.2.

For optimization, we have used the SGD optimizer with a learning rate of 0.01, momentum of 0.8, weight decay of 0.0005, and Nesterov acceleration. To further improve the performance of the model, we have incorporated the ReduceLROnPlateau scheduler with a factor of 0.1 and a patience of 5.

The model's performance has been evaluated through multiple experiments, and the finalized configuration has shown outstanding results. With the identified hyperparameters, our model has achieved an accuracy of 93% on the CIFAR-10 dataset. These findings suggest that the ZigZag ResNet architecture, along with the selected hyperparameters, is a powerful tool for image classification tasks on the CIFAR-10 dataset.

References

- https://github.com/kuangliu/pytorch-cifar
- https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/
- https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6
- https://www.analyticsvidhya.com/blog/2021/08/all-you-need-to-know-about-skip-connections/
- https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/
- https://towardsdatascience.com/a-visual-guide-to-learning-rate-schedulers-in-pytorch-24bbb262c863
- https://www.analyticsvidhya.com/blog/2021/06/understanding-resnet-and-analyzing-various-models-on-the-cifar-10-dataset/