An Modified Residual Network for CIFAR-10 Image Classification

Haoyu Wang(hw3256)*, Yuchen Kou(yk2493)*, Zonghui Hu(zh2393)*

Overview

In this project, we implemented a residual network (ResNet) deep learning model in Pytorch from scratch. The architecture of this network aimed at enabling large amounts of convolutional layers to function efficiently. The ResNet neural networks introduced the concept of skip connections. The idea is to connect the input of a layer directly to the output of a layer after skipping a few connections. This allows us to create deeper networks without worrying about vanishing gradients, which is a common issue for deep neural networks. We use ResNet-18 in this project, which is a 72-layer architecture with 18 deep layers, to perform image classification on CIFAR-10 image dataset.

Summary

To make our model converges fast and classifies images accurately, we experimented with different hyperparameters in the training, such as learning rate, batch size, optimizer, and so on. Comparing each result, we found several combinations of hyperparameters that make ResNet-18 realize more than 90% accuracy, and our optimal ResNet-18 model can be found in this Github Repository[1]. One thing to note, optimizers have the largest impact in the process of model training, and Stochastic Gradient Descent (SGD) achieves highest accuracy of 91% after 120 epochs of training. Detailed explanation will be provided in the following section. In addition, we modified the number of channels in each layer to make the total number of parameters under 5 million. The reduced number of parameters ease the burden of computing and shorten a huge amount of time on training.

Methodology

Our ResNet-18 structure was reproduced from Kuangliu's Github repository, which includes excellent PyTorch code for training various ResNet models on CIFAR-10 from Scratch (Kuangliu 2017). To make the model converge faster and easier for training, we reduced the number of channels to 32 in the first residual layer, 64 in the second residual layer, 128 and 256 in the third and fourth residual layer respectively.

To make up for the accuracy loss in reducing the number of parameters, we experimented the model with different hyperparameters, such as, learning rate, optimizer, batch size and drop out rate. In an effort to analyze our model in each experiment, we made use of torchsummary to output details of each epoch in the training phase, such as training time, training accuracy, test accuracy, and so on, and stored them in a CSV file.

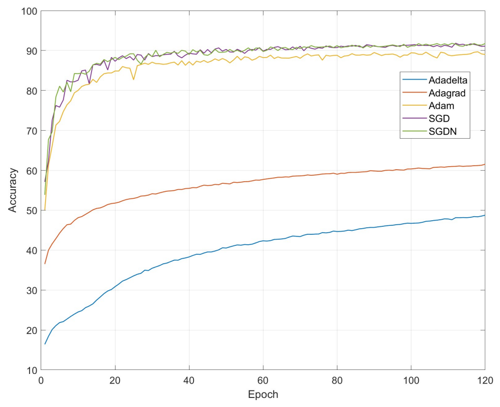

Figure 1: Performance of the model with different optimizers.

.

Optimizer

We firstly tested the performance of the model with different optimizers, while keeping the other parameters constant (Learning rate=1e-4, batch size=128, drop out rate = 0.2). Figure 1 shows the performance of the model with different optimizers. From the graph we can see that SGD and SGD with Nesterov momentum (SGDN) performed similar test accuracy and both achieved 90% after 120 epochs as the intuition behind them are the same, and the only difference is that SGDN adds a correction factor to the standard method, and converges faster in some circumstances. SGDN, however, is more sensitive to the initial value of the learning rate. Adam didn't achieve 90% and fluctuated around 89%

*These authors contributed equally.

[1]https://github.com/HW0327/Building-ResNet-18-for-CIFAR-10-Image-Classification

when it comes to 120 epochs. Adagrad and Adadelta didn't achieve 90% either, and are still increasing after 120 epochs, but we decided to drop them to save training time.

Batch size

When it comes to experiment with batch size, we tested 32, 64, 128, 256 and 512 respectively and set the optimizer as SGD, drop out rate as 0.2, and learning rate as 1e-4. From figure 2 we can see that all kinds of batch size achieved 90% after 120 epochs, and the smaller the batch size, the higher accuracy it has. However, the effect is not significant in our model, so we tested the training time of each batch size. In figure 3, we plotted the average training time per epoch for each batch size, and learned that the smaller the batch size, the longer the training will take. After comparing each test case, we decided to move forward with batch size of 128 as it converges fast while requiring less training time.

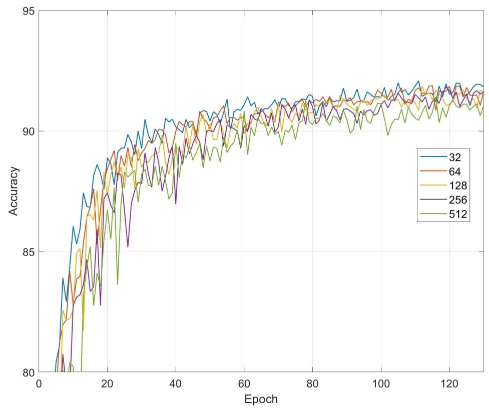

Figure 2: Test accuracy of different batch size.
.

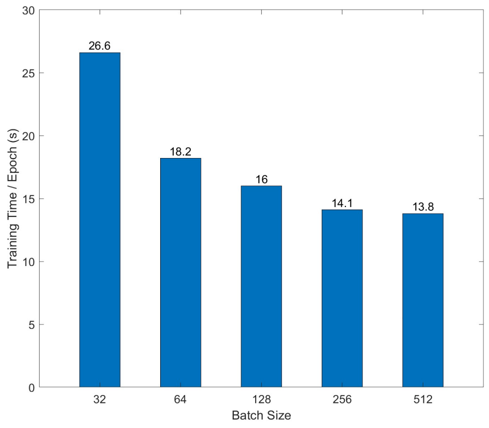

Figure 3: Training time per epoch with different batch size.
.

Learning rate

Learning rate is also an important hyperparameter for the model. Generally, a large learning rate allows the model to learn faster, at the cost of arriving on a sub-optimal final set of weights. A smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take longer to train (Jason Brownlee 2019). we experimented with six different learning rates, from 1e-1 to 1e-6. In figure 4, we can see that with a learning rate of 1e-4, the model converges fastest and approaches the highest test accuracy after 200 epochs.

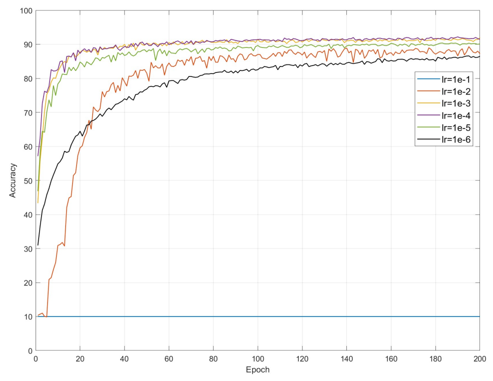

Figure 4: Test accuracy of different learning rate.
.

Drop out rate

The last hyperparameter we experimented on was drop out rate. From the results shown in Figure 5, we can see that there's little impact on the accuracy for different drop out rates. Therefore, we implemented a drop out rate of 0.2 to prevent overfitting, break inter-unit relations and made our model focus more on generalization.

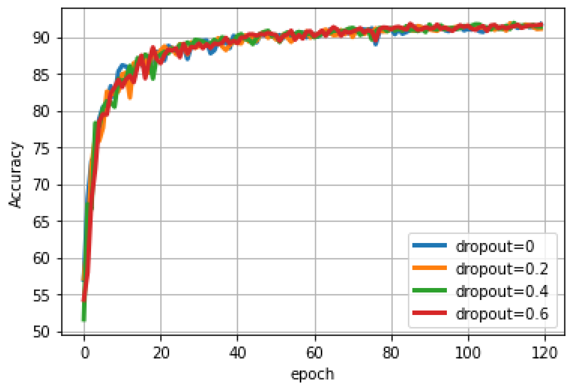

Figure 5: Test accuracy of different drop out rate.
.

Result

Model architecture and Parameters

After the modification of the number of channels in each layer, the number of parameters shrinks to **2,797,610**. Table 1 below is the specific architecture of our model.

Layer (type:depth-idx)	Output shape	Param #
ResNet: 1-1	[-1, 10]	–
–Conv2d: 2-1	[-1, 32, 32, 32]	864
–BatchNorm2d: 2-2	[-1, 32, 32, 32]	64
–Sequential: 2-3	[-1, 32, 32, 32]	–
—-BasicBlock: 3-1	[-1, 32, 32, 32]	18,560
—-BasicBlock: 3-2	[-1, 32, 32, 32]	18,560
–Sequential: 2-4	[-1, 64, 16, 16]	–
—-BasicBlock: 3-3	[-1, 64, 16, 16]	57,728
—-BasicBlock: 3-4	[-1, 64, 16, 16]	73,984
–Sequential: 2-5	[-1, 128, 8, 8]	–
—-BasicBlock: 3-5	[-1, 128, 8, 8]	230,144
—-BasicBlock: 3-6	[-1, 128, 8, 8]	295,424
–Sequential: 2-6	[-1, 256, 4, 4]	–
—-BasicBlock: 3-7	[-1, 256, 4, 4]	919,040
—-BasicBlock: 3-8	[-1, 256, 4, 4]	1,180,672
–Dropout: 2-7	[-1, 256]	–
–Linear: 2-8	[-1, 10]	2,570

Table 1: Modified ResNet-18 model architecture

Final Accuracy

From Table 2, we can find the detailed test accuracy for divergent combinations of hyperparameters. Optimizer has the most significant effect on the performance. And as the key value of an optimizer, abnormal learning rate would influence the accuracy as well. In contrast, different batch size and drop out rate do not bring contrasting performance. The optimal combination of hyperparameters is shown in Table 3.

Hyperparameters	Optimal value
Optimizer	SGD
Learning ate	0.0001
Batch size	32
Dropout rate	0.2

Table 3: Optimal Hyperparameters

Conclusion

This paper shows a modified ResNet-18 architecture with reduced number of channels. Based on the experiment of hyperparameters, we obtained an optimal model. On CIFAR-10 dataset, The model with our optimal hyperparameters has test accuracy higher than 90%.

Along with the emergence of new optimizer, many different combinations and experiments have been left for the future due to lack of time.

References

Jason Brownlee. 2019. How to Configure the Learning Rate When Training Deep Learning Neural Networks. https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/. Accessed: 2022-11-19.

	Optimizer	Learning rate	Batch size	drop out rate	test_avg_loss	test_acc
Control optimizer	AdaDelta	0.0001	128	0.2	1.391	48.77
	Adagrad	0.0001	128	0.2	1.087	61.53
	Adam	0.0001	128	0.2	0.507	89.7
	SGDN	0.0001	128	0.2	0.469	91.81
	SGD	0.0001	128	0.2	0.471	91.85
Control dropout rate	SGD	0.0001	128	0.2	0.471	91.85
	SGD	0.0001	128	0.4	0.436	91.9
	SGD	0.0001	128	0.6	0.48	91.67
Control learning rate	SGD	0.1	128	0.2	9.293	10
	SGD	0.01	128	0.2	0.416	88.38
	SGD	0.001	128	0.2	0.476	91.48
	SGD	0.0001	128	0.2	0.471	91.85
	SGD	0.00001	128	0.2	0.529	90.39
	SGD	0.000001	128	0.2	0.482	84.71
Control batch size	SGD	0.0001	32	0.2	0.409	92.07
	SGD	0.0001	64	0.2	0.421	91.88
	SGD	0.0001	128	0.2	0.471	91.85
	SGD	0.0001	256	0.2	0.489	91.82
	SGD	0.0001	512	0.2	0.507	91.41

Table 2: Final test accuracy

Kuangliu. 2017. Train CIFAR10 with PyTorch. https://github.com/kuangliu/pytorch-cifar. Accessed: 2022-11-19.

Siddharth M. 2021. Understanding ResNet and analyzing various models on the CIFAR-10 dataset. https://www.analyticsvidhya.com/blog/2021/06/understanding-resnet-and-analyzing-various-models-on-the-cifar-10-dataset/. Accessed: 2022-11-19.