Deep Learning Mini Project

https://github.com/Shamitbh/Deep-Learning-Mini-Project.git

Shamit Bhatia[1], Leuber Leuterio[2], and Pragnavi Ravuluri Sai Durga[3]

[1]sb8208@nyu.edu
[2]ll4407@nyu.edu
[3]pr2370@nyu.edu

Abstract

We modified the Resnet-18 architecture, a residual neural network that is 18 layers deep and contains upwards of 11 million parameters. We produced a modified architecture that contains less than 5 million parameters to keep in line with the project requirements. We experimented with various other hyperparamters, optimizers, regularization and data augmentation strategies on our model. In the end, we were able to achieve a 94.6 % test accuracy on the CIFAR-10 image classification dataset.

Introduction

Deep learning has revolutionized the field of computer vision in recent years. Convolutional Neural Networks (CNNs) have proven to be highly effective in tasks such as image classification, object detection, and image segmentation. However, one of the major limitations of these models is their large size, which requires significant computational resources to train and deploy [2]. Hence, there is a growing interest in developing smaller and more efficient CNN architectures while maintaining high accuracy.The Resnet-18 architecture is a deep residual neural network that has been widely used for image classification tasks due to its strong representation power. In this project, we aimed to modify the Resnet-18 architecture to reduce the number of parameters to less than 5 million while maintaining a high test accuracy on the CIFAR-10 dataset. We experimented with various hyperparameters, optimizers, regularization, and data augmentation strategies to achieve our goal.
The remainder of this report is organized as follows. Section 2 provides a detailed methodology of our approach, including the modifications made to the ResNet-18 architecture and the hyperparameter tuning process. Section 3 presents the experimental results and analysis. Finally, section 4 concludes the report with a discussion of the findings and future directions of this work.

Methodology

Larger CNNs have stronger representation power and contain more parameters. This is because theoretically, a neural network with a sufficient large number of parameters can approximate any complex function with any accuracy. One of the limitations imposed was to create a modified residual network with at most 5 million parameters. As such, we aimed to maximize the total number of trainable parameters and achieve at least a 90 percent test accuracy, while also not exceeding the 5 million parameter limit.

We kept the following into consideration when choosing the architecture for our model: We experimented with the number of residual layers, the number of residual blocks in residual layer i, the number of channels in residual layer i, convolutional kernel size in residual layer i, skip connection kernel size in residual layer i. We studied how modifying each individual hyperparameter affected the total number of trainable parameters and test performance. Additionally, we analyzed how different optimizers, learning rate schedulers, and data augmentation strategies affected test performance.

Training

All Models were trained, validated, and tested using the same platform through Jupyter notebook in Google Colab. This platform allows users to prototype machine-learning models on Graphical Processing Units (GPUs).

Additionally, all models conducted image classifcation on the CIFAR-10 dataset. CIFAR-10 is an established computer-vision dataset used for object recognition. The dataset consists of 60,000 32x32 color images with three channels, containing one of 10 object classes, with 6000 images per class.

Each image of the dataset was normalized such that each pixel value was in the range of [0,1]. The dataset was then split into training, validation, and test sets. We used a validation set of 5000 images (roughly 10% of the dataset).

During the training phase of each experiment, the training and validation loss, as well as the accuracy, were recorded at the end of each epoch. Cross-entropy loss was used as our loss function. The model with the lowest validation loss across all epochs has its parameters saved into a state dictionary. During the evaluation phase, the parameters of the model with the best validation loss is then loaded from the state dictionary and evaluated on the test dataset to come up with a test accuracy for that model.

Optimizer and Learning Rate Scheduler

We limited our project to exploring three popular optimizers: We compared the results of SGD assisted with momentum, ADAM, and ADAMW. The total number of training epochs is 100 and the learning rate is 1e-3.

Optimizer	Test Accuracy
SGD + Momentum	71.6
ADAMW	76.26
ADAM	84.3

In the absence of data augmentation, ADAM achieved the best results. For all experiments, we used the ADAM algorithm [3] an optimizer to adapt the learning rate for each weight. In addition, we improved the behavior of the learning rate on each epoch during the training using the technique of reducing-learning-rate-on-plateau. This technique improved accuracy, descending into areas of lower loss by monitoring the loss during the training. The learning rate would be reduced if the loss was stagnant for several epochs.

Data Augmentation Strategy

We used the following data augmentation strategy during training. We performed Random Rotation, where the image is rotated randomly within a range of 5 degrees. We performed Random Horizontal flip that randomly flips the image along the horizontal axis. We performed Random Crop, where 2 pixels are padded on each side, and a 32×32 crop is randomly sampled from the padded image.

We also used a procedure available in PyTorch called AutoAugment. AutoAugment has a CIFAR-10 policy that performs mostly color-based transformations that are listed in the table below [1].

	Operation 1	Operation 2
Sub-policy 0	(Invert,0.1,7)	(Contrast,0.2,6)
Sub-policy 1	(Rotate,0.7,2)	(TranslateX,0.3,9)
Sub-policy 2	(Sharpness,0.8,1)	(Sharpness,0.9,3)
Sub-policy 3	(ShearY,0.5,8)	(TranslateY,0.7,9)
Sub-policy 4	(AutoContrast,0.5,8)	(Equalize,0.9,2)
Sub-policy 5	(ShearY,0.2,7)	(Posterize,0.3,7)
Sub-policy 6	(Color,0.4,3)	(Brightness,0.6,7)
Sub-policy 7	(Sharpness,0.3,9)	(Brightness,0.7,9)
Sub-policy 8	(Equalize,0.6,5)	(Equalize,0.5,1)
Sub-policy 9	(Contrast,0.6,7)	(Sharpness,0.6,5)
Sub-policy 10	(Color,0.7,7)	(TranslateX,0.5,8)
Sub-policy 11	(Equalize,0.3,7)	(AutoContrast,0.4,8)
Sub-policy 12	(TranslateY,0.4,3)	(Sharpness,0.2,6)
Sub-policy 13	(Brightness,0.9,6)	(Color,0.2,8)
Sub-policy 14	(Solarize,0.5,2)	(Invert,0.0,3)
Sub-policy 15	(Equalize,0.2,0)	(AutoContrast,0.6,0)
Sub-policy 16	(Equalize,0.2,8)	(Equalize,0.6,4)
Sub-policy 17	(Color,0.9,9)	(Equalize,0.6,6)
Sub-policy 18	(AutoContrast,0.8,4)	(Solarize,0.2,8)
Sub-policy 19	(Brightness,0.1,3)	(Color,0.7,0)
Sub-policy 20	(Solarize,0.4,5)	(AutoContrast,0.9,3)
Sub-policy 21	(TranslateY,0.9,9)	(TranslateY,0.7,9)
Sub-policy 22	(AutoContrast,0.9,2)	(Solarize,0.8,3)
Sub-policy 23	(Equalize,0.8,8)	(Invert,0.1,3)
Sub-policy 24	(TranslateY,0.7,9)	(AutoContrast,0.9,1)

Application of AutoAugment signifcantly improved test performance compared to just using the previous listed data augmentation strategies.

For validation and testing, we only evaluated the single view of the original 32×32 image.

Number of Channels in Residual Layer *i*

The number of channels in each residual block of ResNet-18 is [64, 128, 256, 512]. This yields 11.1 million trainable parameters. We halved the number of channels in each layer, which resulted in [32, 62, 128, 256] channels in each residual block. We used this as a baseline and analyzed how increasing the number of channels in each layer affected the total number of trainable parameters and also test performance.

Channels	Parameters(M)
32, 64, 128, 256	2.8
64, 64, 128, 256	2.9
32, 128, 128, 256	3.2
32, 64, 256, 256	4.5
32, 64, 128, 372	4.9

In each case, increasing the number of channels in residual layer *i* resulted in an increase in test performance, when compared to baseline. The increased performance between each architecture, other than baseline, was not significantly different.

In particular, we observed that increasing the channel size in residual layers 3 and 4 resulted in consuming a significant number of our 5 million parameter budget. This would limit other modifications to the network that we could implement. As a result, we avoided increasing the channel size in residual layers 3 and 4.

Filter size across all Residual Layers

With baseline channel sizes = [32,64,128,256], we observed that increasing the filter size to k = 4 across all residual layers resulted in an increase in the number of parameters with no gains in test accuracy. Further increases in the kernel size across all residual layers resulted in an excess of 5 million parameters. A kernel size of k=3 was chosen across all residual layers.

Skip Connection Kernel size across all Residual Layers

With baseline channel sizes, we modified the skip connection kernel size across all residual layers. When analyzing filter sizes between k=1 and k=3, a filter size of k=3 resulted in optimal test accuracy.

Filter Size	Parameters(M)	Test Accuracy
1	2.8	78.65
2	2.9	77.48
3	3.8	81.21

Residual Block Sizes

We did not observe any obvious correlation between block sizes and test accuracy. Due to the training time and number of possible combinations, we limited training to 40 epochs and tested with Channels = [64,128,128,256]. Also, increasing block sizes in earlier stages resulted in smaller increases in the number of parameters compared to increasing block sizes in later stages. We chose to focus on increasing more block sizes in earlier stages to allow for more fine-tuning when considering other hyperparameters.

Block Sizes	Parameters(M)	Test Accuracy
[2,2,2,2]	3.1	91.6
[3,2,2,2]	3.4	90.77
[3,3,2,2]	3.7	91.55
[3,3,3,2]	4.0	91.61
[4,3,3,2]	4.1	92.90
[5,4,3,2]	4.5	91.44
[8,4,3,2]	4.7	90.99

Models

All experiments were trained with the following hyperparameters:

epochs = 150
batch size = 64
Conv. kernel size = 3
Skip connection kernel size = 3
Average pool kernel size = 4
optimizer = ADAM
Scheduler = Reduce on Plateau

With the exception of model 4, all models were trained using the specified data augmentation strategies. Model 4 is the same as model 1, without the application of data augmentation. In model 5, an extra residual layer was added, which corresponds to the 5 different block sizes. Model 6 is the same as Model 1, except SGD with momentum = 0.9 was used as the optimizer

Model	Channels	Block sizes	(M)	Acc.
1	[64, 128, 128, 256]	[5,4,3,2]	4.95	94.63
2	[32, 64, 128, 256]	[10,8,6,2]	4.91	94.39
3	[32, 128, 128, 256]	[10,4,4,2]	4.99	94.10
4	[64, 128, 128, 256]	[5,4,3,2]	4.95	84.02
5	[32, 64,128, 128, 256]	[5,4,4,3,2]	4.97	93.77
6	[32, 64,128, 128, 256]	[5,4,4,3,2]	4.97	93.79

Training Loss, Validation Loss for each Model

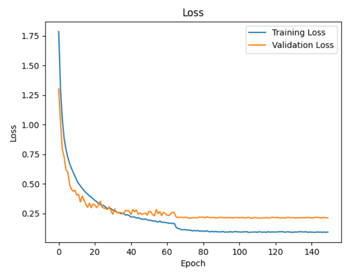

Figure 1: Model 1 had the highest test accuracy of 94.63

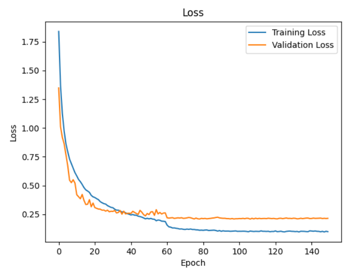

Figure 2: Model 2 has smaller channels and more blocks compared to Model 1

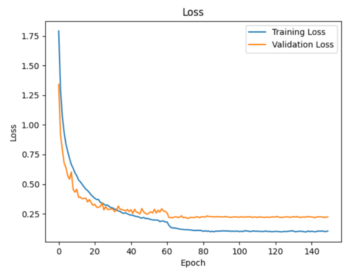

Figure 3: Model 3 has smaller first channel and more blocks compared to Model 1

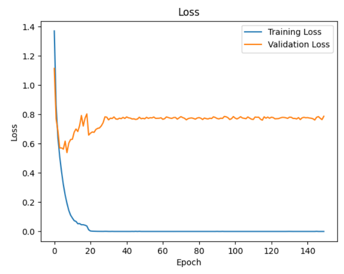

Figure 4: Model 4, same as Model 1 but no augmentation

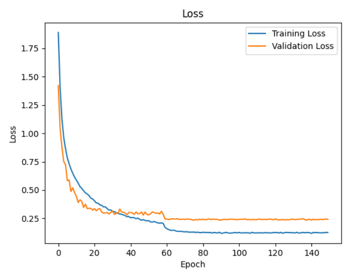

Figure 5: Model 5 has an extra residual layer added

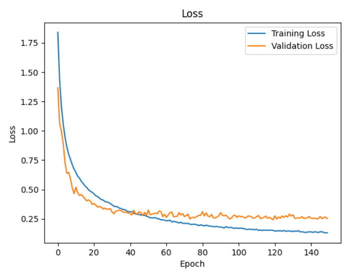

Figure 6: Model 6 is the same as Model 1 but with SGD momentum=0.9

The experiments indicate that while Model 1 had the highest test accuracy of 94.63, Models 2 and 3 also had similar performances with accuracies above 94 percent. Model 2 may be more efficient since it consumes less parameters (4.91 M) when compared to Model 1 (4.95 M), while having a test accuracy of 94.39.

It is apparent that data augmentation plays a more significant role in test performance improvements than when any of the structural changes were applied. This is evident from Model 4, when the model clearly overfits the training data in the absence of data augmentation. While data augmentation does improve test performance drastically, it does so at a cost. The training time for Model 4 is 13 sec per epoch, where as the training time for the other models was 20-21 sec per epoch.

Furthermore, we did not see gains in test accuracy when adding a 5th residual layer (Model 5) nor did we see gains in test accuracy when using SGD with momentum as the optimizer (Model 6).

Results

Model Description

For our finalized model, we used the following data augmentation strategy: RandomRotation, RandomHorizontalFlip, RandomCrop, and Autoaugment on CIFAR-10 policy. We trained the model for 150 epochs, using a batch size of 64, cross entropy loss as our loss function, ADAM as the optimizer, and Reduce on Plateau a the learning rate scheduler.

The architecture of the model starts with a convolutional layer (2D convolutional operation + 2D batch norm) followed by four residual layers. The kernel size in each residual layer = 3x3. The number of channels in each residual layer = [64, 128, 128, 256]. The number of basic blocks in each residual layer = [5, 4, 3, 2]. Each residual block contains a skip connection with a kernel filter size = 3x3. The residual residual layers are then followed by an average pool layer with kernel size = 4x4, a fully connected layer, and a softmax.

Final Model Architecture

Layer name	Model 1
Conv1	3x3, 32, stride = 1
Residual 1	[3x3, 64] [3x3, 64] x 5, stride = 1
Residual 2	[3x3, 128] [3x3, 128] x 4, stride = 2
Residual 3	[3x3, 128] [3x3, 128] x 3, stride = 2
Residual 4	[3x3, 256] [3x3, 256] x 2, stride = 2
	4x4 Average Pool
	FC
	Softmax
Skip Connection	3x3, stride = 1
Total Parameters	4.95 million
Final Test Accuracy	94.63

Discussion

Our final model was able to achieve 94.6% accuracy on the CIFAR-10 dataset using 4.95 million parameters. While, our aim was to achieve the highest test accuracy through maximizing the number of parameters allowed under 5 million, we did not consider efficiency when designing our model. Further studies could be undertaken to modify the architecture to be more parameter efficient. More exploration is needed to assess whether each layer is needed. Eliminating unproductive layers would make the architecture more lightweight, efficient, and would require less training.

References

[1] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical data augmentation with no separate search. *CoRR*, abs/1909.13719, 2019.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.