Modified residual network (ResNet) architectures for CIFAR10 image classification

Geetika Bandlamudi(gb2642)*, Ramanarayanan Sankaranarayanan (rs8117)*, Vinayak Muttathu Santhoshkumar (vm2368)*

GitHub: Link

Abstract

This report consists of the experimentation and our findings of using a Modified ResNet architecture to classify CIFAR-10 image dataset into 10 classes. The highest accuracy achieved by our architecture is 93.06%. We have discussed about the different architectures along with the details about which approach works and why.

Introduction

We are tasked to build a modified residual network (ResNet) architecture with the highest test accuracy on the CIFAR10 image classification dataset, under the constraint that our model has no more than 5 million parameters.

The main constraint in this task is that the designed ResNet architecture should have no more than 5 million trainable parameters. Hence, we have architectured our model and verified the same using torchsummer.summary function. The ResNet architecture consists of convolutional layers with skipped connections, where each block implements ReLU(S(x) + F(x)), where S(x) refers to the skipped connection and F(x) is a block that implements conv $->$ BatchNorm $->$ relu $->$ conv $->$ BatchNorm. We have experimented with modifying the hyperparameters such as the number of channels, filter size, kernel size, and pool size, and experimented with optimizers, regularization techniques, learning rates, batch sizes, and epochs along with data augmentation strategies to achieve the best possible test accuracy. We did not use any other additional datasets.

Methodology

1. Baseline Architecture

Starting from the ResNet's fundamental architecture as shown in [2], we shall proceed. With four residual layers and channel sizes ranging from 64,128,256, and 512, it has 11.17 million parameters. Training this model for 100 iterations, resulted in a test accuracy of 86.8% using Cosine Annealing for learning rate scheduler with an initial learning rate of 0.1 and SGD. We want to alter ResNet's architectural design so that there are fewer than 5 million parameters, but we also want to experiment with different learning rates and optimizers to see if we can match or even surpass the test accuracy we obtained with the baseline architecture. We modified the Kernal size to 3 from 1 and added a padding of 1. We also included an average pooling layer, which is a parameter-free operation, to compress the image size.

2. Architecture Changes

We started off with modifying the channel sizes in the baseline architecture of ResNet with 4 residual layers. We used He Normal initialization for the convolutional layers and Xavier Normal initialization for the fully connected layer. We also initialized the batch normalization layers with a scale factor of 1 and a bias term of 0. We also used Adam optimizer, ran it for 100 epochs and a learning rate of 0.001. However, we observed a test accuracy of 84.69%. We expected a slightly higher accuracy though, since the highest validation accuracy was around 88.25%. The total number of parameters in this model was 4,366,250 which is way less than 5M. Our goal was to experiment with the choice of learning rate, optimizer to achieve or even improve upon the test accuracy we got for this architecture. Here, we have presented the following alternate models and discussed about how each decision had an impact on the test accuracy.

2.1 Modified Architecture

Here, we removed the parameter initialization of the model since it was not improving the accuracy. We believe this is because the weights are initialized to very large or very small values, the gradients can become too large or too small and lead to poor convergence. Similarly, if biases are initialized to non-zero values, it can shift the activation function and cause poor convergence. This architecture has a fixed input size of 3x32x32 (3 color channels, 32x32 spatial dimensions). Each residual block contains two convolutional layers, each followed by batch normalization and a ReLU activation function. The output of the second convolutional layer in each block is added element-wise to the output of the first convolutional layer, before passing through another ReLU activation function. The first residual block has 32 filters and is followed by two additional residual blocks with 32 filters. Then, there are two residual blocks with 64 filters, and each block has two convolutional layers followed by batch normalization and ReLU activation. This architecture has a total of 4,366,250 parameters and achieved a test accuracy of 90.46%. We used ReduceLROnPlateau Learning

*These authors contributed equally.

Rate Management with an initial learning rate of 0.1 which is explained in detail in the next section. We noticed that the learning rate changed once. Even in this model, Adam was used. We later experimented with different optimizers as explained in Section 4 and decided to proceed with SGD for further improvement of accuracy.

2.2. Deeper Network with Same stride

Since the ResNet has an advantage of its ability to train very large number of parameters with no substantial increase in the training error percentage, we wanted to explore how a deeper network would perform. We experimented with the channel sizes and we ended up with a deeper network with 8 residual layers of sizes 16, 32, 64, 96, 128, 156, 192, 220 to stay within 5M parameters. All the layers have a stride of 2 except the first. We used SGD as an optimizer with ReduceLROnPlateau as the scheduler for 100 epochs. This model gave a test accuracy of 91.46%. This is a higher accuracy compared to the previous model. The deeper network has more parameters and learnt more complex representations of the input data. This in-turn led to better generalization and higher accuracy. The different channel sizes allowed for more diversity in the features that are learned by the network and hence resulted in feature extraction and improved accuracy.

2.3 Deeper Network with Alternating Stride

As a modification to this model, we made sure that this model has a stride size of 2 for every alternating residual layer only. We used SGD as an optimizer with ReduceLROnPlateau as the scheduler for 100 epochs. This model gave a testing accuracy of 89.19%. This is not better than the above two discussed models. One reason why there is a lower accuracy is because alternating stride sizes can lead to a decrease in spatial resolution, which may cause loss of information and accuracy. Also, one other argument can be that alternating stride may lead to gradient vanishing or exploding, as the gradient flows through several layers with different stride sizes leading to slow convergence and suboptimal solutions.

2.4 Shallower Network

One other experimentation we did was to work with less number of trainable parameters because we noticed that the deeper network was taking a lot of training time. This new architecture has 3 residual layers with sizes 64, 156, 220. this resulted in a total of 2.5M parameters. To compensate for the loss of trainable parameters, we experimented with a starting learning rate of 0.1 and incorporated the ReduceLROnPlateau learning parameter scheduler strategy. Hence, we expected this approach to converge at an optimal result quickly and trained this model for 100 epochs as well. We did not increase the number of epochs as the training error was close to zero at around 0.070 by just 100 epochs. This gave the highest test accuracy of 93.06%. By using a smaller number of parameters, the model was less prone to over fitting and was able to generalize better to unseen data. It can be safely concluded this combination of model architecture, training strategy, and hyperparameters led to a high-performing model with a lower number of trainable parameters, making it an effective solution classifying CIFAR-10 images.

3. Learning Rate Management

While monitoring the train and validation accuracy, we utilized a different scheduler from the one used in the initial architecture called ReduceLROnPlateau. When the designated measure stops improving for a longer period of time than the patience number permits, ReduceLROnPlateau is a scheduling strategy that lowers the learning rate. As a result, the learning rate is maintained at its current level for as long as it improves the metric amount, but it is decreased when the results begin to plateau. Since there is no improvement after a predetermined number of epochs, we divided the learning rate by 10. We then compare the performances over 100 training epochs with the initial learning rates set to 0.1 for all the methods.

We observed that the learning rate was divided twice while training a Deeper network with same stride, once while training a deeper network with alternating stride, once while training a shallower network. It is also noteworthy that when we started with a learning rate of 10^{-3}, we did not observe a change in the learning rate since the total number of epochs was not enough to spot a plateau. However, since we started with a learning rate of 0.1, we could observe a change in the learning rate as the model plateaued. We have coloured the region where the learning rate changed and marked it accordingly in the graphs.

4. Optimizer changes

4.1 Adam

We started off with Adam and later made a choice to proceed with SGD. As we know, Adam is the first immediate choice of an optimizer due to its robustness and ease of use. But Adam can sometimes struggle to converge to an optimal solution, especially in cases where the learning rate is too high or the loss surface is particularly rugged. When we experimented with Adam, with a learning rate of 0.1, we got an accuracy of 90.46%.

4.2 SGD

By using SGD with an appropriate learning rate and learning rate schedule, we were able to achieve a higher accuracy than with Adam. SGD with momentum can help to smooth out the updates and reduce oscillations, especially in cases where the loss surface is rugged or the gradients are noisy. Additionally, SGD with momentum has fewer hyperparameters to tune compared to Adam, which can simplify the training process.

4.3 Adagrad

The Adagrad optimizer maintains a per-parameter learning rate that is scaled inversely proportional to the square root of the sum of the squares of the gradients for that parameter. This means that the learning rate decreases over time as the optimizer accumulates more and more gradients. We used Adagrad on the initial architecture with a batchsize 128. It resulted in a Test accuracy of 85.52% and a test loss of 0.449.

4.4 RMSProp

RMSProp is a gradient-based optimization algorithm that is similar to Adagrad, but with a modification to address its tendency to reduce the learning rate too quickly for deep neural networks. The key difference between RMSprop and Adagrad is that RMSprop uses a moving average of the squared gradient instead of accumulating all the historical squared gradients. RMSProp resulted in a Test accuracy of 89.69% and a Test loss of 0.420.

Result and Metrics

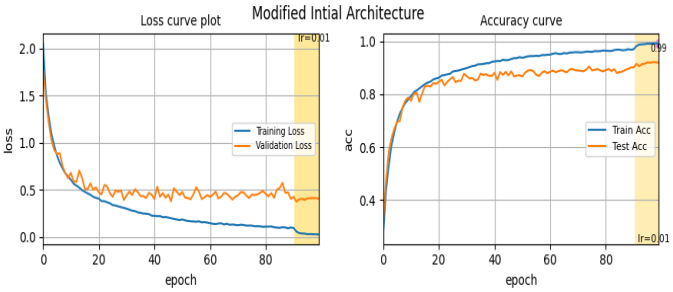

Figure 1: Modified Initial Architecture

Model	Training Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
Initial Architecture	92.53	88.25	84.69
Modified Initial Architecture	99.17	91.89	90.46
Modified Initial With SGD	98.13	92.77	92.38
Modified Initial With Adagrad	99.42	89.80	85.52
Modified Initial With RMSProp	96.87	90.57	89.69
Deeper with same stride	97.86	90.18	89.19
Deeper with Alternating stride	99.38	92.44	91.46
Shallow Network	97.59	93.28	93.06

Table 1

Model	Number of Parameters	Colab Link
Initial Architecture	4.36M	Link
Modified Initial Architecture	4.36M	Link
Modified Initial With SGD	4.36M	Link
Modified Initial With Adagrad	4.36M	Link
Modified Initial With RMSProp	4.36M	Link
Deeper with same stride	4.95M	Link
Deeper with Alternating stride	4.95M	Link
Shallow Network	2.56M	Link

Table 2

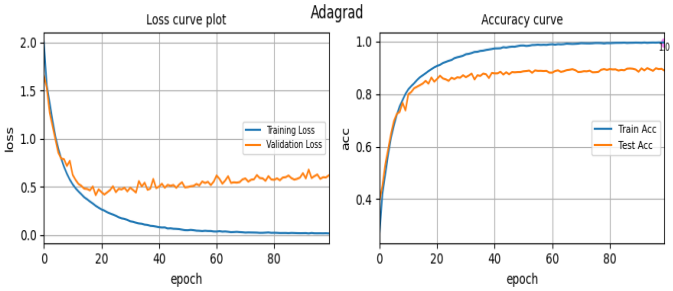

Figure 2: Using Adagrad Optimizer

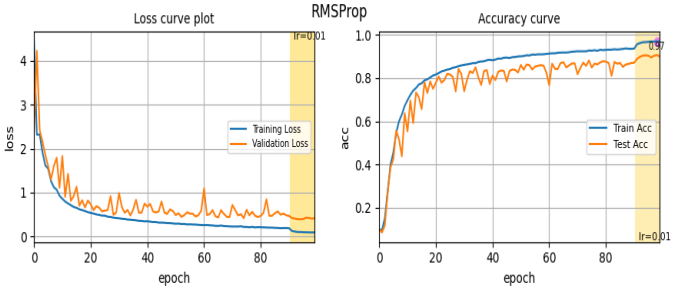

Figure 3: Using RMSProp Optimizer

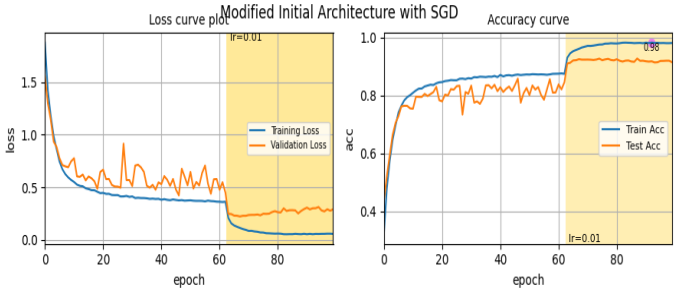

Figure 4: Using SGD Optimizer

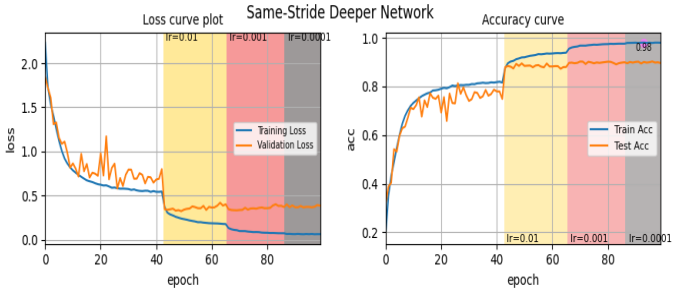

Figure 5: Deeper Network with Same Stride

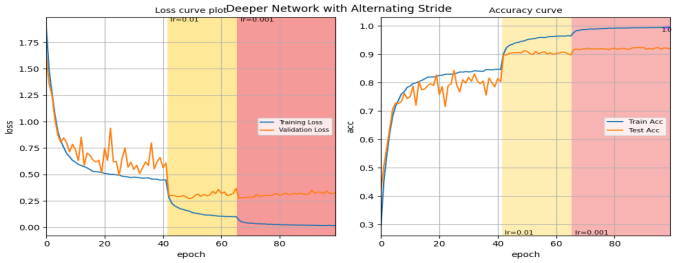

Figure 6: Deeper Network with Alternating Stride

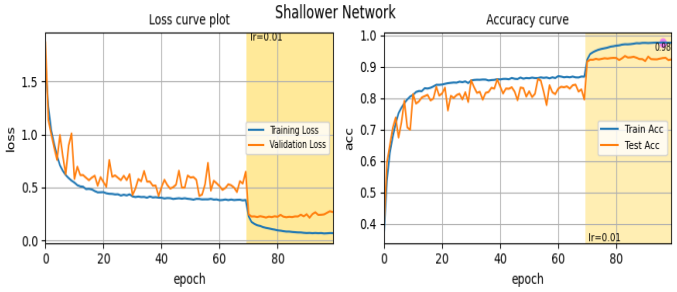

Figure 7: Shallower Network

Conclusion

The model that provided us the highest accuracy (93.06%) out of all the experiments and methods we tried was the Shallow Network model. One thing we saw in all of our studies was that smaller models worked better. This, in our opinion, is caused by the tiny dataset and the few classes. By limiting the amount of parameters to 5 million, we were able to achieve this great precision. This discovery may possibly be due to the shallower model's superior ability to generalize and the dense model's tendency to overfit to the training set. Additionally, we observed that SGD optimizer outperforms Adam, RMSProp, and AdaGrad optimizzers in terms of performance.

References

[1] Kmldas, "Cifar10 resnet: 90 plus % accuracy;less than 5 min," Jan 2021

[2] kuangliu, *Train CIFAR10 with PyTorch*, https://github.com/kuangliu/pytorch-cifar.

[3] Priya Goyal, Piotr Doll´ar, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. arXiv preprint arXiv:1706.02677, 2017.

[4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. "Going deeper with convolutions." IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015. https://arxiv.org/abs/1409.4842