Mini Project 1: Residual Network Design

Kewal Jani, Karan Parikh, Ahmad F Ishraq
New York University
{kj2062, kap9580, afi225} @ nyu.edu

Abstract

With more people gaining access to limited-memory devices, there has been a growing need to adapt the complexity and depth of existing deep neural network models as per the storage limitations of these devices. One approach to take account of such limitations is to reduce the number of trainable parameters in existing models. In our work, we extended on the existing ResNet-18 model by reducing the model parameters below 5 million while maximizing the accuracy benchmark on the CIFAR- 10 dataset.

1 Introduction

Recent advances in deep neural networks [1,2] have achieved numerous state of the art results in image classification tasks. It was found that even though deeper networks [3] yield breakthrough results [1], they are faced with a degradation problem [3]. As the network gets deeper, accuracy was observed to fall substantially as the gradient kept diminishing with depth. This problem wasn't due to overfitting since training loss and test loss was observed to be deteriorating. Residual Networks [3] have introduced the concept of a skip connection that allows the network to resolve the vanishing gradient problem by adding back the input of the current block to its output. This network architecture has substantially improved performance [3] over previously available convolutional networks. It was also shown that with deeper layers, these networks performed much better. In our project, we were provided with the ResNet-18 architecture that was used in [3] and we will be presenting our ResNet architecture with lower than 5M parameters compared to the original network.

The ResNet-18 architecture provided to us consists of around 11.17M trainable parameters. In order to achieve our goal of adapting this model to be stored on devices with limited storage capacity, we had to specifically look into ways of reducing our model size to contain parameters under 5M whilst also maintaining good accuracy. This made us take a new approach of using various block size combinations/configurations within each residual layer along with optimization techniques. We then show our findings by training our optimized model on the CIFAR-10 train dataset and testing it on the CIFAR-10 test dataset.

2 Methodology

In this section, we go over the various methodologies we employed to implement our ResNet architecture.

The original ResNet-18 architecture[1] provided to us had the following configurations as shown in Fig. 1.

To start off, we decided on a modest batch size of 64 images for both the training and testing sets. CIFAR-10 provides 50000 images in the training set and 10000 images in the test set.
In our initial approach we decided to tease the idea of removing the last layer completely. Before

[1] by original we are referring to the document provided to us.

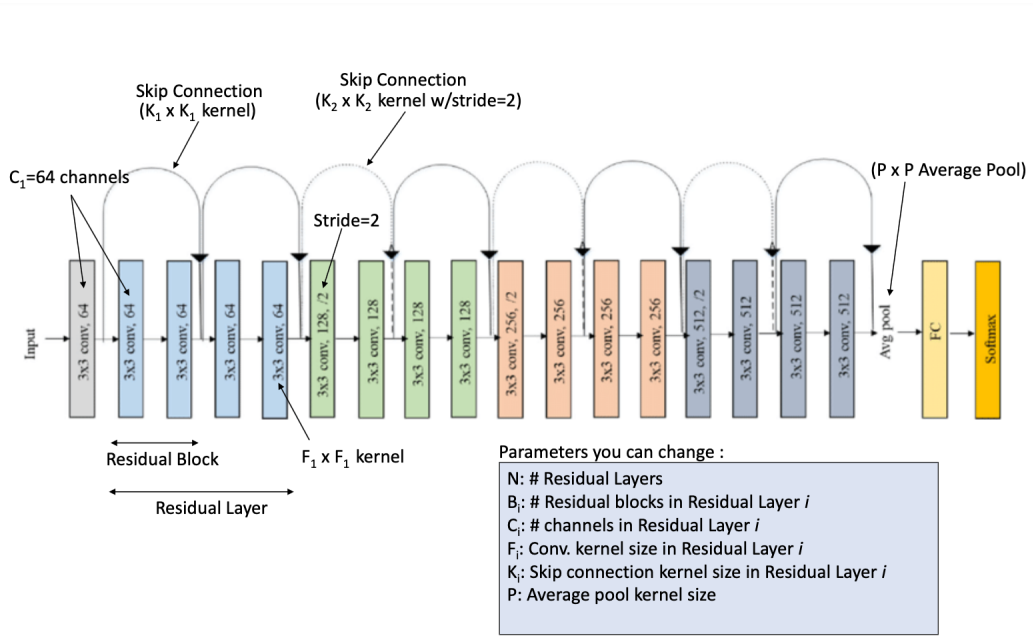

Figure 1: Original ResNet-18 Architecture.

Block Configuration	**Paramaters**
2-2-2-2	11,173,962
2-2-3-1	7,633,994
2-2-2-1	6,453,322
2-2-1-1	5,272,650

Table 1: Block size configuration rejected due to exceeding parameters

doing so, we tested various block configurations with 4 layers to see if any of those configurations fit within our constraint of 5M parameters. The results are mentioned in Table 1. The original layers with 2-2-2-2 block size had around 11.1M parameters and reducing one block in the last layer(2-2-2-1) still gave us more than 5M parameters. Only 2 configurations stood out: 2-1-1-1 and 1-1-1-1. Both these configurations had parameters below 5M but their training and testing accuracy did not seem very promising after running 10 epochs. You can observe these results in Table 2. In the case of 1-1-1-1 block size the accuracy was 86.31% in the test case at 10 epochs. Block sizes of 2-1-1-1 gave us more than 80% train and test accuracy but higher loss. Further reducing blocks in the first layer gave us better accuracy but the loss did not improve. This led us to omit the last layer entirely instead of compromising on the number of blocks in the first 3 layers of our network.

On removing the last layer entirely, from the original architecture we were left with B_i=[2,2,2] for i=1,2,3. We conducted training for 2-2-2 for 10 epochs, and got train and test accuracy in the range of 60-65%. In order to improve on this, we tested different block configurations possible for 3 layers with each configuration running for 10 epochs. The results can be observed in Table 3. We chose 10 epochs given the limited resources, most importantly training time, that was available to us. Our objective here was to find the combination that yielded highest accuracy and the lowest loss on both train and test sets. We started with blocks 1-1-1, 2-3-3 and 3-3-3. In all instances we observed that accuracy increased as the number of blocks in each layer increased. Conversely, loss was also observed to be decreasing. This led us to try an extreme combination of 7-4-3. Results for this combination did not look promising.

Our current best try was 2-3-3 configuration. It had achieved the highest accuracy till now

Block Configuration	Parameters	Train Accuracy	Test Accuracy	Train Loss	Test Loss
2-1-1-1	4977226	0.8177	0.826	0.527	0.503
1-1-1-1	4900682	0.86566	0.8631	0.388	0.3967

Table 2: Valid 4-Layer Block size configuration results after 10 epochs

(Optimizer used: Stochastic Gradient Descent(SGD) with momentum=0.9)

Block Configuration	Train Accuracy	Test Accuracy	Train Loss	Test Loss	Optimizer
1-1-1	0.7498	0.7575	0.7158	0.6969	SGD
2-3-3	0.8878	0.8785	0.3285	0.3588.	ADAM
7-4-3	0.8694	0.8552	0.3744	0.4275	SGD
7-4-3	0.0994	0.100	2.3026	2.3025	ADAMW
3-3-3	0.8857	0.8740	0.3304	0.3839	ADAM
4-5-3	0.8702	0.8603	0.3770	0.4145	SGD
5-4-3	0.8869	0.8747	0.3261	0.3690	SGD

Table 3: 3-Layer Block size configuration results after 10 epochs

which was 87.85% . Next, we tested combination 5-4-3, which was marginally close in terms of accuracy (87%+) and loss (36%). So we decided to run both of these for 100 epochs on our train and test sets to see if one improved over the other when the number of epochs were much higher. Thus, we set 5-4-3 as our final block configuration. Removing the third layer and only working with 2 layers would not make any sense since the deeper layers with 256 channels are more important than the first few layers with 64 or 128 channels because they can capture features much more prominently.

2.1 Network Architecture

Based on our experiments in the previous section, we have decided to use the following configurations: (see Figure 2)

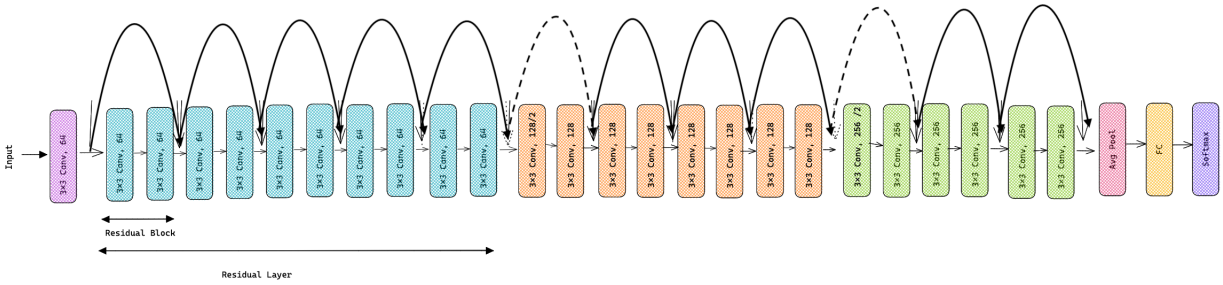

Figure 2: Our ResNet Architecture

Residual Layer. The model consists of N=3 Residual Layers. Layer i has B_i blocks (for our network, B_i = [5,4,3] for i = 1, 2, 3). The first layer has five residual blocks, followed by four in the next layer and 3 in the last layer. Residual layers are preceded by a fixed convolutional block and followed by an average pooling and fully connected layer.

Residual Block. Our residual block contains two convolutional layers with a skip connection from the block's input to the block's output, similar to original architecture. Similarly, we consider implementing the following sequence of operations: conv→bn→relu→conv→bn→(skip connection)→relu (bn is batch normalization).

A residual block implements:

$$relu(F(x) + conv(x)),$$

where the $conv(x)$ term corresponds to the skip connection.

Input Layers. The 32 × 32 × 3 input image is processed by conv layer 3 × 3 sized filters and C_1 output channels. This is followed bn and relu.

Average Pool and Output Layers. The last residual layer's output feeds into a 3 × 3 average pooling layer with 8 channels, followed by an appropriately sized fully connected layer. Finally, the output of dimension 10 is passed through a softmax layer.

In Fig. 2 we show our final network architecture.

2.2 Optimization Techniques:

We also applied various optimization techniques on the final block configuration to improve the accuracy of our model even further. They are listed below:

2.3 Testing with different optimizers

We ran our 5-4-3 block configuration using 5 different optimizers - SGD Nesterov, SGD(Momentum=0.9), ADAM, Adagrad and ADAMAX. The results that we obtained are as follows:

Optimizer	Train Accuracy	Test Accuracy
SGD(Nesterov)	99.09%	91.13%
ADAM	98.304%	91.2%
ADAMAX	98.962%	91.26%
Adagrad	98.96%	89.91%
SGD(Momentum)	99.68%	91.45%

Table 4: Accuracies when testing with different optimizers for 5-4-3 block configuration

We decided to go ahead with SGD(momentum=0.9)(91.45%) and try other optimization or data augmentation techniques to improve the accuracy of our model.

2.3.1 Network Dropout

We also tested regularization methods such as using dropouts in our 5-4-3 network. We observed a degradation of performance. In comparison to the results in Table 2, we obtained train and test accuracy at 71.2% and 73.5% respectively for the same configuration for 5-4-3 block configuration for about 10 epochs.

2.3.2 Data Augmentation

CIFAR (Canadian Institute For Advanced Research) has prepared a dataset of 10 classes: Airplane, Automobile, Cat, Deer, Dog, Frog, Horse, Ship, Truck and Bird. The images are 32*32 pixels in size. The training set consists of 50000 images, with 5000 of each class. The test set consists of 10000 images with 1000 for each class. In our experiment, we use the train set, augment it and then calculate accuracy and loss on the validation (or test) set.

For Augmentation we used the following methodologies :

In this implementation we only use horizontal flips (flipped 50% of training data-set). We pad the images into size 40x40 using reflective padding and then crop the images back into size 32x32.Random cropping is used as an augmentation in the training phase. To normalize the data we found the mean and standard deviation of the RGB channels and got the values as following and used it to Normalize the data set.

Mean : 0.49139968 0.48215827 0.44653124
Standard deviation : 0.24703233 0.24348505 0.26158768

Why did we use these augmentation techniques?

We found the following article[4] which states the optimum settings and best augmentation techniques for CIFAR-10. On analyzing our result(shown below) we applied the best settings for our ResNet model.

3 Results

We ran the training and testing for 100,120 and 150 epochs. The results were as follows.
We got 92.96% for 100 epochs, 93.09% for 120 epochs and **93.11% for 150 epochs** respectively.

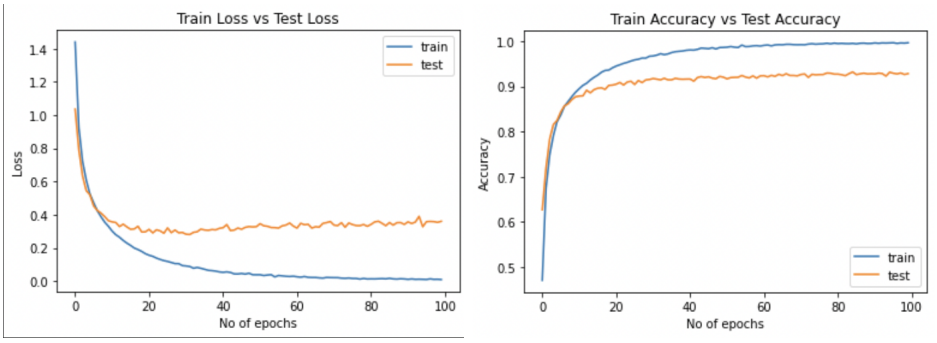

Figure 3: Train and test Accuracy and Loss over 100 epochs

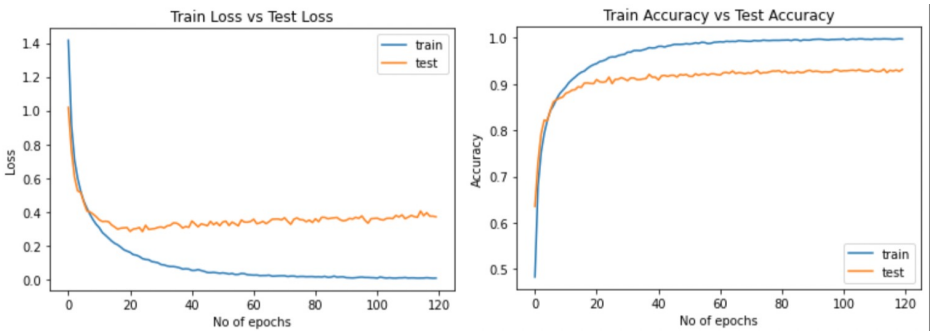

Figure 4: Train and test Accuracy and Loss over 120 epochs

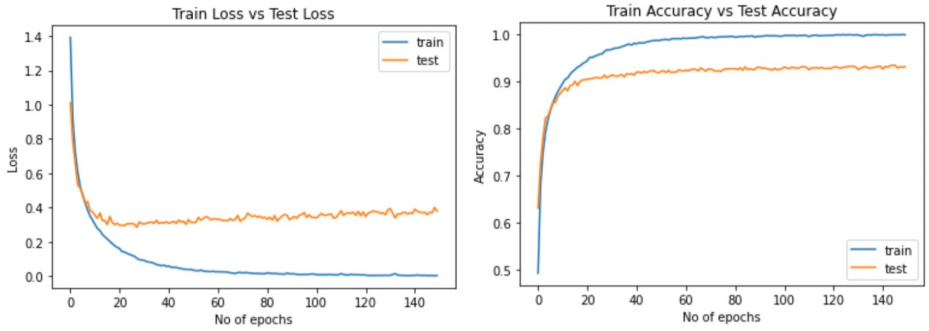

Figure 5: Train and test Accuracy and Loss over 150 epochs

The total number of parameters are: **4,771,146**
The test error is : 6.89%

4 Conclusion

We managed to increase the accuracy of our model within the constraint of 5M parameters using various architectural, optimization and data augmentation techniques. Further improvements can be done by adding the 4th layer of 512 channels as well. We even implemented the above idea using factorization of convolution by breaking down the 3x3 convolutions in every layer using 1x1 bottleneck convolutions. This technique helps to reduce parameters by a third thus allowing us to add a fourth layer. We achieved a base accuracy of 93% and we believe the accuracy can increase further using data augmentation or optimization techniques.

5 References

[1] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In arXiv:1512.03385 [cs.CV], 2015.

[4]https://medium.com/swlh/how-data-augmentation-improves-your-cnn-performance-an-experiment-in-pytorch-and-torchvision-e5fb36d038fb.

[5] https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363

[6]https://bytepawn.com/solving-cifar-10-with-pytorch-and-skl.html

[7]https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

[8]K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.

[9]A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.

Github Repo Link : `https://github.com/kewaljani/Resnet_5Mpara`