Mini-Project 1: Residual Network Design

Feng Yun
ECE, NYU Tandon
fy2054@nyu.edu

Youming Lu
ECE, NYU Tandon
yl8025@nyu.edu

Aurora Stan
ECE, NYU Tandon
zz3015@nyu.edu

Abstract

As obstacles are exposed when using Convolutional Neural Networks(ConvNets) with tens of layers, ResNet architecture models are designed. In this project, to increase the accuracy on the CIFAR-10 dataset, a 10-layer ResNet model is designed. The main structure is that every two convolutional layers is considered as one residual block and one skip connection is designed between every residual block. The input of the next residual block is the sum of the input and output of the last cycle. To artificially expand the dataset, random cropping and random flipping were used in the data augmentation process. Adam is chosen as an optimizer to get results more efficiently compared to SGD. After training the ResNet model with 50 epochs, we get a test accuracy at 90.84%. Our code and ResNet model are available at `https://github.com/mirevas/DeepLearning.git`

1 Introduction

As the problem of vanishing/exploding gradients expose when stacking more layers to the neural networks, normalized initialization and intermediate normalization layers are the most accepted ways to addressed this problem, which enable networks with tens of layers to start converging for stochastic gradient descent(SGD) with back propagation. [1] However, degradation problem arisen, where the accuracy gets saturated and degrades rapidly with the depth increasing. This non overfitting problem is unexpected and even get worse in training error when adding more layers according to articles[2 3]. A deep residual learning framework has come up.

We established the ResNet model with 10 layers as Fig. 4. It contains 4 residual layers in total, includes one residual block every single residual layer and holds 2 blocks every single residual block. Before entering the residual layers, one fixed convolutional block is executed. An average pooling and a fully connected layer implemented at the end of all the residual layers.

In this paper, it demonstrates the methodology specifically, including introducing the residual learning, choosing the proper optimizer(Adam), modifying the residual layer and preparing the data set. The results are produced by plotting the training and test loss and the corresponding accuracy graphs.

2 Methodology

2.1 Residual Learning

The depth of the deep learning network is critical to the performance of the model. From experience, as the number of network layers is increased, the network can extract more complex feature patterns, and theoretically better results can be achieved when the model is deeper. However, as the depth of the network increases, the accuracy of the training set saturates or even declines, which is so-called degradation problem, as shown in Fig. 1[1].

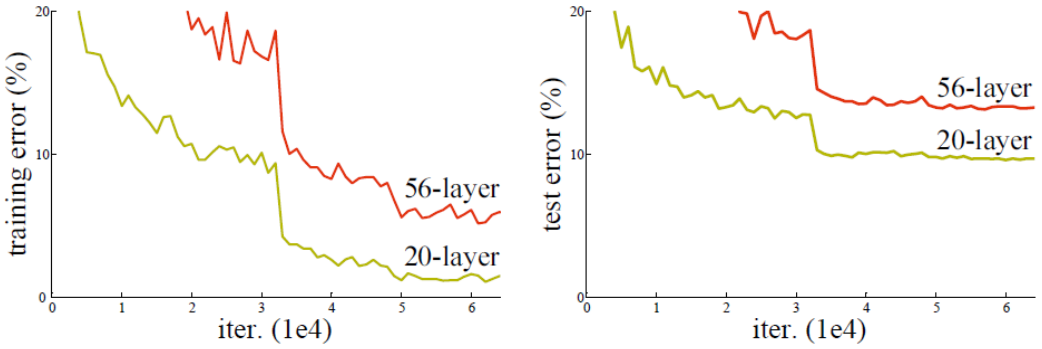


Figure 1: Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks.

The degradation problem is not due to overfitting, because the training set should have high accuracy by overfitting. We know that deep networks suffer from vanishing or exploding gradients, which makes deep learning models difficult to train. But there are already some technical means such as Batch-Norm to alleviate this problem. Therefore, it is surprising that the degradation problem still occurs in the deep networks.

To solve this problem, the Resnet structure was proposed in 2015[1]. As shown in Fig. 2, Resnet learns the residual mapping F(x) through identity mapping (Skip Connection). In this case, if the network has reached the optimal state and continue to deepen the network, the residual mapping will be pushed to 0, leaving only the identity mapping, therefore, theoretically, the network has always been in an optimal state, and the performance of the network will not decrease with the depth increases.

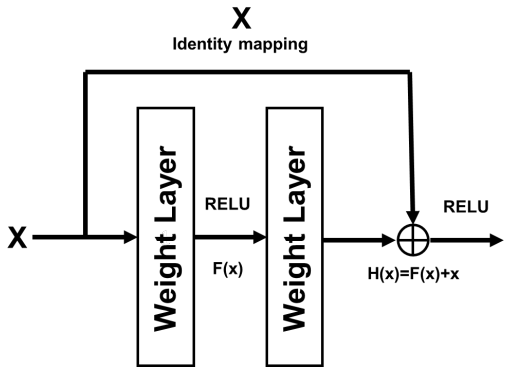


Figure 2: Residual Learning

2.2 Network Architecture

In order to make the various parameters of the model compound requirements, we made some adjustments below:

Optimizer Choice. Stochastic Gradient Descent (SGD) is a simple but very effective method, which is mostly used for the learning of linear classifiers under convex loss functions such as support vector machines and logistic regression, and SGD has been successfully applied in frequently encountered large-scale and sparse machine learning problems in text classification and natural language processing. From this, optimization algorithms such as SGD with Momentum (SGDM) and SGD with Nesterov-Acceleration are derived.

Although the basic mini-batch SGD optimization algorithm has achieved many good results in deep learning, there are still some problems to be solved.

SGD relies on manual parameter tuning to obtain better optimization results, but it is difficult to choose an appropriate initial learning rate, and the learning rate adjustment strategy is limited by pre-specified adjustment rules, and the same learning rate is also applied to each parameter. In particular, the biggest disadvantage of SGD is that the descending speed is slow, and it may continue to oscillate on both sides of the gully, staying at a local optimum.

On this basis, we choose the Adam optimizer, which was proposed by Kingma and Lei Ba in 2015[4]. The parameter updates of the Adam optimizer are not affected by the scaling transformation of the gradient, the hyperparameters are well interpretable, and usually require little or no tuning.

Residual Layer. The typical Resnet structure proposed by KaiMing He is shown in Fig. 3[1]. However, even the most concise resnet18 architecture still contains about 11M parameters, as shown in Fig. 4(b).

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3: Typical Resnet Architectures

Therefore, we try to modify the corresponding network parameters to meet the parameter size requirement of less than 5M. Considering the basic structure of Resnet and the necessity of skip connection, we reduced the number of 4 residual blocks from 2 to 1 on the basis of resnet18.

The revised Resnet architecture is shown in Fig. 4(a). The revised Resnet has 4 residual blocks, each has 2 convolution layers with kernel size of 3, and the input channel size of first layer of first block is 64. Besides, on the basis of skip connection, if the input channel size of latter layer is equal to the output channel size of previous layer, the part of identity mapping does not need to be convolved, so K1 should be 0. As the convolution goes forward, the skip connection meets the output size of 64 and the input size of 128, the identity mapping part needs to be convolved and we set the kernel size K2 as 3. Finally, the kernel size of average pool has been set to 4. We can also see in Fig. 4(c) that through the revision, the total number of parameters in the network model has been reduced from 11M to 4.9M.

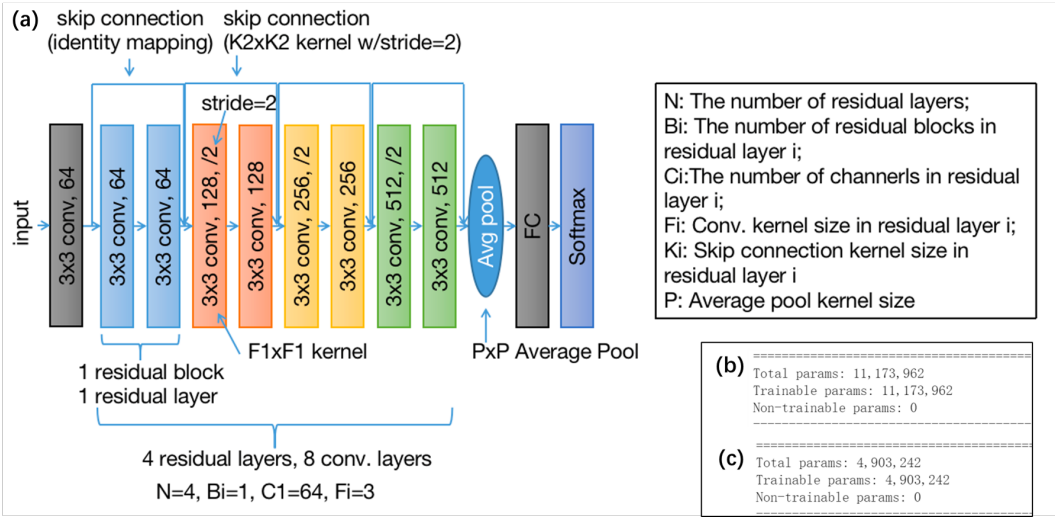


Figure 4: (a) Revised Resnet architecture. (b) Total parameter quantity of Resnet18. (c) Total parameter quantity of Revised Resnet

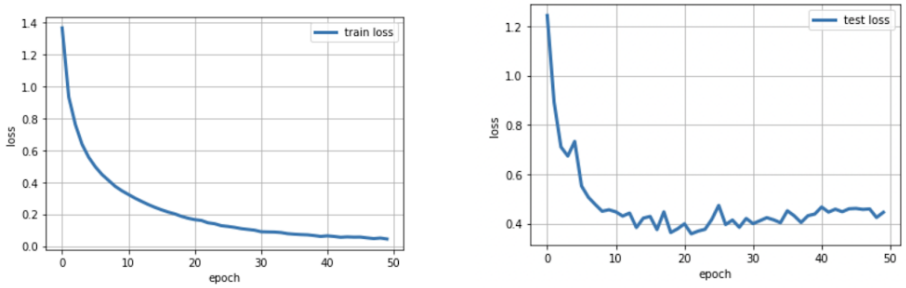
Data Pre-treatment. To make the model more robust and make predictions more accurate, we need to preprocess the data. The dataset we used is CIFAR-10, which has a total of 60,000 labeled color images of size 32*32, divided into 10 classes with 6,000 images per class. There are 50,000 images for training, 5,000 images per class, and another 10,000 images for testing, 1,000 images per class.

To make the model more robust, we first randomly cropped each image. Using the Pytorch function *transforms.RandomCrop()* , fill 0 around the margin, and then randomly crop the image into 32*32. Then use the Pytorch function *transforms.RandomHorizontalFlip()* to horizontally flip the cropped image with probability p (default as 50%). Convert the processed image to a tensor using the Pytorch function *transforms.ToTensor()*, and then do normalization using the Pytorch function *transforms.Normalize()*. Here we choose [0.4914, 0.4822, 0.4465] and [0.2023, 0.1994, 0.2010] as the normalize parameters.

3 Results

The final architecture we use is that there are 4 residual layers, the number of residual blocks in each residual layer are 1,1,1,1. The total number of trainable parameters of this architecture is 4,903,242.

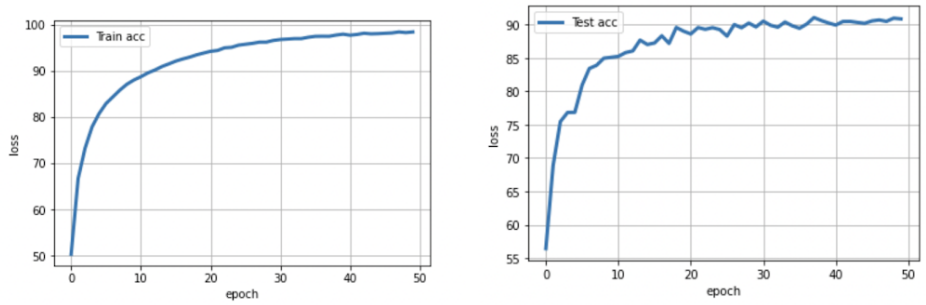
The curves plotting the training and test loss are below:



(a) Train Loss (b) Test Loss

Figure 5: Plots of training and test loss

The curves plotting the training and test accuracy are below:



(a) Train Accuracy (b) Test Accuracy

Figure 6: Plots of training and test accuracy

From the above experimental results, we can see the train loss and test loss become very small. Moreover, the train accuracy is 98.38% and test accuracy is 90.84%. The train and test accuracy are above 90%, which means there is no over-fitting. The number of parameters required is less than 5M in this project. So, for further work, it is very meaningful to use more complex models to improve accuracy. In addition, preventing overfitting is also very important.

4 Conclusion

In this work, we applied the resnet18 as the prototype and to change a part of the architecture of this model. Because the parameter quantity is required to be less than 5M. We have 4 residual layers and each residual layer has only one residual block. In each residual block, we have the structure like conv→batchnorm→relu→conv→batchnorm→(maybe shortcut). We also use some data augmentation strategies like picture crop and flip. The CrossEntropy is chosen as a loss function. Adam is the optimizer in this model. Then we use CIFAR-10 as the dataset to train the model by running 50 epochs. We finally got the test accuracy 90.84%.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[2] Kaiming He, Jian Sun. Convolutional neural networks at constrained time cost. In CVPR, 2015.

[3] Rupesh Kumar Srivastava, Klaus Greff, Jürgen Schmidhuber. Highway networks. arXiv:1505.00387, 2015.

[4] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In International Conference on Learning Representations (ICLR), 2015.