Optimization of Residual Network with given budget

Authors
Yiwei Zhang (yz7303)
Yuyang Liu (yl8019)
Dachuan Zuo (dz2231)

Abstract

This work mainly focus on the optimization of **Residual Network** with a given architecture. The overall goal is to maximize the accuracy with a limited trainable parameters of **5 million** over the **CIFAR-10** database. The final model achieved above 90% accuracy.

1 Introduction

The given Residual Network is a Wild RESNET with a widen factor $k = 2$. With the limitation of 5 million As the requirement of remain the architecture unchanged, our optimization is composed by the following five parts:

1.1 Hyper-parameter

The changeable hyper-parameters include: residual layer N, residual block number B_i, first residual layer channel number C_1, input filter size F, residual block filter size K, and average pooling filter size P. Among all the hyper-parameters, only the average pooling filter size P does not effect the number of trainable parameter. Therefore, it will be discussed separately.

1.2 Data Augmentation Strategies

We survey the accuracy of different combination of data augmentation strategies over the same hyper-parameter. The experimental data will be detailed examined in the **Methodology** section.

1.3 Optimizer

We choose ADAM as optimizer of our model, because it is outperform SGD.

1.4 Regularization Scheme

A small weight decay was add to ADAM as regularization.

1.5 Training Procedures

We use 64 batch size and 50 epoch during experiment period, and applied the dynamic learn rate in our model.

2 Methodology

The provided example ResNet-18 contains about 11 million trainable parameters. This number is more than double of the given budget of 5 million. Thus a decrease of trainable parameters number is required: the channel number of first residual layer is cut from 64 to 32 in order to decrease the number of trainable parameters to around 2.8 million. This model will be used as a baseline for accuracy comparison, cause its trainable parameter number is about half of the given budget.

Throughout the paper, we will use the following denotation rule to represent models, using the baseline model as example:

$$(N = 4, B = [2, 2, 2, 2], C_1 = 32, F = 3, K = 3, P = 4)$$

And, for simplicity, the hyper-parameter in any model will be omitted if same as the baseline model.

The optimization process is guided by the **control variable** experimental method. The accuracy of each optimizing factor will be compared with others under a relatively same situation for each part. By default, accuracy given in this report is chosen as the median of 5 runs. Otherwise, the detail of accuracy chosen will be stated.

The sequence of the five optimization parts written in this report is not the actual sequence in our work. The training procedures was determined first. Second, ADAM is chosen as the optimizer because it outperforms SGD. After that, we add regularization to our model. Then, data augmentation was applied. Finally, we conduct the experiment will different combination of hyper-parameter to find the optimal one.

2.1 Hyper-parameter

Because the trainable parameter number is dependent on all hyper-parameters except the average pooling filter size, a certain range of trainable parameter number should be keep same when compare the combination of hyper-parameters. Therefore, the optimization of hyper-parameter is divided into the following three steps.

2.1.1 Model Complexity

The first step of optimizing hyper-parameters is to determine an optimal model complexity. As the number of trainable parameters is directly related to the model complexity, a range of trainable parameters number should be selected before starting tuning other hyper-parameters.

Considering the given budget of 5 million trainable parameters is relatively small, we raise the hypothesis that the optimal range of trainable parameters should close to the limitation because of the full utilization of model complex. Based on the hypothesis, four different model are tested: the baseline model, $(N = 3, B = [2, 2, 2])$ (one layer less than the baseline model), $(C_1 = 16)$ (half first channel), and $(B = [3, 3, 3, 3])$ (almost reach the budget limit). The data is showed in **Table 1**.

Table 1: Accuracy over different trainable parameter number

MODEL	PARAMETER NUM	ACCURACY
(B=[2,2,2,2],C=16)	701466	84.37%
(B=[2,2,2,2],C=32)	2797610	85.47%
(B=[2,2,2,2],C=40)	4368690	85.33%
(B=[3,3,3,3],C=32)	4366250	85.75%

A tendency of positive relation between the trainable parameter number and accuracy can be find with **Table 1**. Therefore, our hypothesis is confirmed. Besides, when we continue add the output channel number of the first convolutional layer after 32, the accuracy start decrease. Thus we decided that $C = 32$ is an optimal value.

2.1.2 Average Pooling Filter Size

The number of trainable parameters is independent from average pooling filter size P, so it is tested separately with other hyper-parameters. Limited by the image size of **CIFAR-10** database and the

4 layer model, 4 is the maximum. Thus the average pooling filter size P has a range of $[2, 4]$. It is tested over the baseline model. The result is showed in **Table 2**.

Table 2: Accuracy with different pooling filter size

pooling filter size	**ACCURACY**
2	82.76%
3	83.26%
4	83.32%

Based on the test result, only little difference on accuracy of different average pooling filter size is observed. So we decide to keep the default value of average pooling filter size $P = 4$ as this value is slightly better than the other two.

2.1.3 Other Hyper-parameter

The original paper of wild residual network suggest to use 3 as filter size in residual blocks[1]. So we decided to keep the residual block filter size $K = 3$ unchanged. Also, 4 layer network can achieve higher accuracy with data augmentation.

We had studied the structure of residual network layers, and find that a mountain shape block distribution (less block in two end of the network and more block in middle) can help improve accuracy[2].

Table 3: Accuracy over different hyper-parameter

MODEL	**PARAMETER NUM**	**ACCURACY**
(1.B=[2,4,8,2],F=3,C=16)	1182490	84.21%
(2.B=[2,4,8,2],F=3,C=32)	4718122	85.04%
(3.B=[2,4,8,2],F=5,C=32)	4719658	84.79%
(4.B=[4,6,8,2],F=3,C=16)	1228954	83.76%
(5.B=[4,6,8,2],F=3,C=32)	4903210	86.15%
(6.B=[4,6,8,2],F=5,C=32)	4904746	85.92%

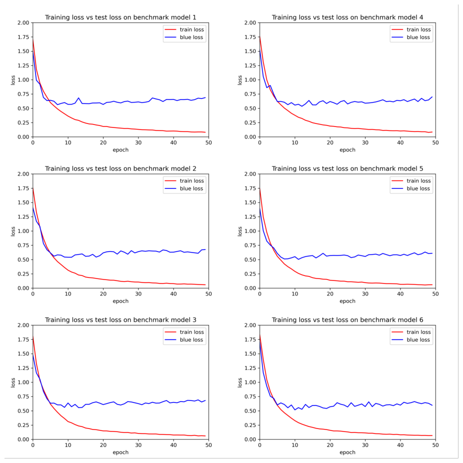

Figure 1: Benchmark loss

Table 3 provides the accuracy of some combination of the undetermined hyper-parameter. We choose the

2.2 Data Augmentation Strategies

Three regular data augmentation strategies are used in our work to reduce overfitting on training data: RandomCorp, RandomFlip and RandomGrayscale. **Table 4** shows the accuracy of baseline model with and without data augmentation. Their train and test loss is in **Figure 1** and **Figure 2**.

Table 4: Accuracy with and without data augmentation

DARA AUGMENTATION	ACCURACY
without	77.84%
with	83.89%

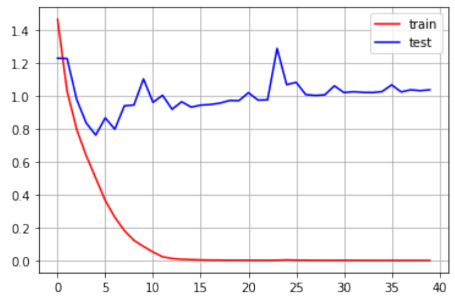

Figure 2: Without data augmentation

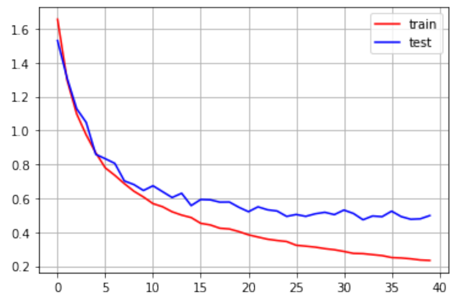

Figure 3: With data augmentation

With these data augmentation, we prevent the overfitting in early epoch and make it possible to train with more data. Beside, data augmentation generalizes our model to fit better on testing set.

2.3 Data-Augmentation: Mixup

Mixup is another data augmentation strategies we applied in our model. It trains a neural network on convex combinations of pairs of examples and their labels. As a result, mixup can regularizes the neural network to favour simple linear behaviour in-between training examples. Experiments have proven that mixup is able to reduce error rate of ResNet when training on **CIFAR-10** database[3].

2.4 Optimizer

ADAM can outperforms SGD while share about same training cost and converge speed[4]. We also conducted our own experiment. **Table 5** shows the comparison of ADAM and SGD over the baseline model. **Figure 3** and **Figure 4** graph the train and test lose for ADAM and SGD separately.

Table 5: Accuracy of ADAM and SGD

OPTIMIZER	ACCURACY
ADAM	87.35%
SGD	84.55%

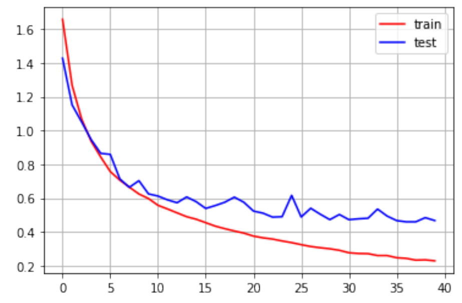

Figure 4: SGD Loss

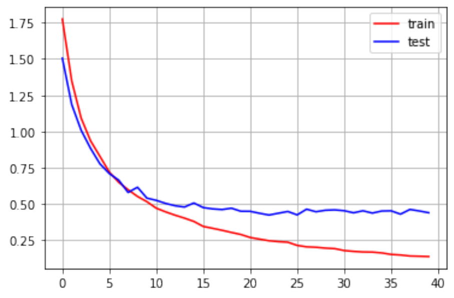

Figure 5: ADAM Loss

With the data, we verify that using ADAM as optimizer will provide a better accuracy with a more smoothing test loss.

2.5 Regularization Scheme

Adding a small weight decay can significantly increase the accuracy compared with other regularization method[5]. Thus, we set the weight dacay to be 1e-9 in ADAM optimizer.

2.6 Training Procedures

Throughout the project, all the models are trained with 64 batch size. This choose is based on two reasons. First, our model is a light-weighted model limiting by the given budget, thus large batch will cause the degradation of the model[6]. Second, the 64 batch, as a small batch will help our model avoid trapping in local optimization.

The epoch number is set to 50 during the experiment period, as most train and test loss is observed to be flatten and accuracy starts to fluctuate around a certain range when epoch reach this point. Therefore, we decide to stop at 50 epoch to save time.

When we run the given example ResNet-18 in the very beginning of our work, a very quick overfitting on the training data is observed. To reduce the overfitting, we applied the dynamic learn rate in our model. We set up a step scheduler to multiply the learning rate with 0.9 for every 4 epoch.

3 Results

This section serves the purpose of demonstrating final architecture of model, providing a description of training process, and showing the results of prediction.

3.1 Network Architecture

Residual Networks offer an excellent balance of performance and parameter count, as well as the training speed. We starts from the ResNet18 model. The ResNet18 model consists of 4 residual blocks and each of the blocks is configured as the sequence of BN-ReLU-Conv-BN-ReLU-Conv. As shown in **figure *reduced vs original.png***, the network first perform convolution and pooling operation before the input enters the residual blocks. The first convolution layer has 64 output channels, a kernel size of 3, and a stride size of 1. The 4 residual blocks change the size of feature map when entered by the input with output channels of 64, 128, 256, and 512. The first block has a stride size of 1, while all of the other 3 have a stride size of 2. An average pooling operation is performed at the end of residual blocks. After that a fully-connected layer of 512 dimension is used to generate predictions.

In order to reduce the complexity of the model, changes are made to keep the number of parameters of the model under 5 million. Our reduced network has 4 residual layers which is the same as the original network. However, the number of blocks under the 4 layers are changed to 4, 6, 8, 2 accordingly. The dimensions of each convolution layer are also cut to half in our reduced model, resulting in output channels of 32, 32, 64, 128, and 256 for the first convolution layer and the 4 residual blocks respectively compared with 64, 64, 128, 256, 512 in the original ResNet18. The kernel sizes and stride sizes of each convolution layer remain the same. The dimension of the fully-connected layer at the end of the network is also changed to 256 to be consistent with the out channel of the last residual block. A comparison of the architecture of our reduced ResNet and the original ResNet18 are shown in **figure *reduced vs original***.

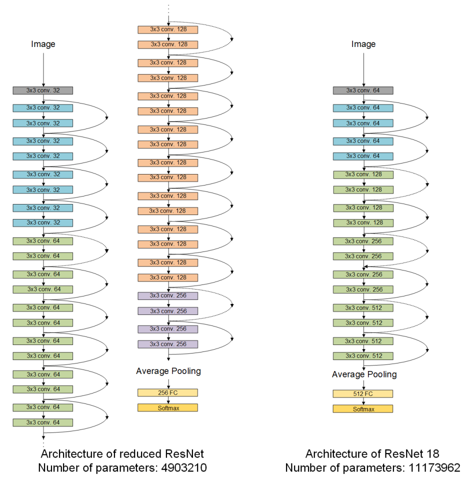
Image
Average Pooling
256 FC
Softmax
Image
Average Pooling
512 FC
Softmax
Architecture of reduced ResNet
Number of parameters: 4903210
Architecture of ResNet 18
Number of parameters: 11173962

Figure 6: Reduced vs Original

We use the well-studied CIFAR10 dataset to train and test our model. The input images are normalized using the per-channel means and standard deviations in the CIFAR10 dataset. Each image input is 32x32x3 in size. The dataset is divided into a training set that is learned by the network and a test set used to validate and test the performance. A sequence of data augmentation operations are performed on the image inputs to achieve better generalization. These operations include random crop, random flip, and random gray scale. Cross-entropy loss is used to compute the loss in predictions.

We choose ADAM as the optimizer with beta 1 and beta 2 set to 0.9 and 0.999, and epsilon set to 1e-8. To attain the effect of L2 regularization, the weight decay value of ADAM is set to 1e-9. To balance the training speed and the performance, we adopt a decayed learning rate that starts from 0.01 and is forced to reduce by 5% every 5 epochs. We have noticed a significant improvement in the training speed after using the decayed learning rate compared to using a constant 0.01 learning rate.

We also employ the Mixup operation with λ = 0.2 to further improve the training accuracy and prevent overfitting. The Mixup operation produces additional samples during the training process that involves convexly blending random pairs of images and their corresponding labels. An appreciable improvements of prediction accuracy on the test dataset was noticed after Mixup was implemented. It is worth noting that the Mixup operation converts the image input and the associated label to $\lambda x_1 + (1-\lambda)x_2$ and $\lambda y_1 + (1-\lambda)y_2$. The associated loss function will be converted to $L(\hat{y}, \lambda y_1 + (1-\lambda)\lambda y_2)$, which uses the mixed images and mixed labels and instead of the original images and labels.

3.2 Loss and Accuracy

figure *reduced resnet loss.png* shows the change of loss on the training and test samples over 300 epochs of training. It is noticeable that the training loss was always higher than the test loss. The training loss after MixUp tends to be higher than any other models we have tested (**figure *benchmark loss.png***). Moreover, the loss curve on the training samples was volatile while decreasing **figure *reduced resnet loss.png*** in our 'mixed-up' training in contrary to the smooth decreasing curves without the Mixup (**figure *benchmark loss.png***). These are because the training losses are computed using the mixed images and labels instead of the original images and labels after the training samples are mixed up.

Another notable finding in our 'mixed-up' training is that the training loss keeps going down and starts to converge after 300 epochs. It is evident in **figure *reduced resnet loss.png*** that the overfitting had little impacts during our 'mixed-up' training as the loss curve on the test samples are still in a downward trend when reaching 300 epochs. In contrast, the model without the Mixup operation tends to overfit after 40 epochs as shown in **figure ?**. These differences suggest the MixUp operation has successfully prevent the overfitting in our training. Our final prediction accuracy on the test dataset falls at 94%, which was a noticeable improvement over the 6 benchmark models in **figure *benchmark loss.png***

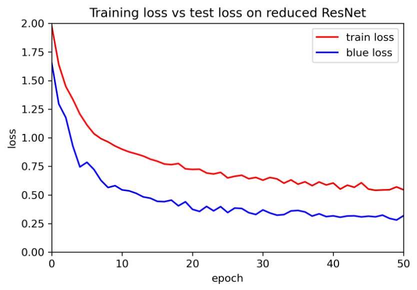

Figure 7: Reduced resnet loss

4 Github Link

Github repository link:
`https://github.com/Nightingalelyy/ECE-7123-DeepLearning-Project1`

Reference

[1] Sergey Zagoruyko & Nikos Komodakis (2016) Wide Residual Networks

[2] Kaiming He & Xiangyu Zhang & Shaoqing Ren & Jian Sun (2015) Deep Residual Learning for Image Recognition. *Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015*

[3] Hongyi Zhang & Moustapha Cisse & Yann N. Dauphin & David Lopez-Paz (2018) mixup: Beyond Empirical Risk Minimization *Published as a conference paper at ICLR 2018*

[4] Diederik P. Kingma & Jimmy Ba (2015) Adam: A Method for Stochastic Optimization.

[5] Irwan Bello & William Fedus & Xianzhi Du & Ekin D. Cubuk & Aravind Srinivas & Tsung-Yi Lin & Jonathon Shlens & Barret Zoph (2021) Revisiting ResNets: Improved Training and Scaling Strategies

[6] Nitish Shirish Keskar & Dheevatsa Mudigere & Jorge Nocedal & Mikhail Smelyanskiy & Ping Tak Peter Tang (2016) On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima