Modified Residual Network for Image Classification

Shuning Li, Ziyi Huang

ECE-GY 7123 Deep Learning, 2024 Spring
Mini Project
Codebase: Modified ResNet on CIFAR-10
sl10916@nyu.edu, zh2931@nyu.edu

Abstract

Residual Network (ResNet) is introduced to address the vanishing/exploding gradient problem and is popular in image classification tasks. In this project, we modified the ResNet-18 architecture to allow the trainable parameter count under 5 million. Our objective is to achieve an optimal model performance on the CIFAR-10 dataset by experimenting, comparing, and contrasting different hyperparameters, optimizers, and data augmentation techniques.

Introduction

Deep neural networks are used to face a significant problem: as the network gets deeper, the training accuracy would initially improve but then saturate and even decline. This problem is known as the vanishing/exploding gradient problem, which made the networks difficult to train as the error rate ceased to decrease, and additional layers did not contribute to improved accuracy. Residual Network (ResNet), a CNN-based architecture is introduced to address this issue in the groundbreaking paper (He et al. 2015). The authors designed a network with building block of the structure shown in Figure 1, where the ”weight layer” blocks are convolutional layers of the same kernel size. The skip connection from the input of the block to the summation sign provides a reference, so that the output of this block can be at least as good as $\mathbf{x}$ itself. If we consider the total mapping to be learned by this block to be $\mathcal{H}(\mathbf{x})$, then the path with convolutional layers and nonlinearity, or $\mathcal{F}(\mathbf{x})$, approximates the residual function $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$. That’s why these blocks are called ”Residual Blocks” or ”Residual Units” as in (He et al. 2016), and how the framework got its name. A number of Residual Blocks with one kernel size are connected in series to form a Residual Layer, and 4 Residual Layers with increasingly sized kernels, together with an Input Layer and an Output Layer form the backbone of ResNet.

The original ResNets were trained on ImageNet. Depending on the different number of total convolutional layers used, the ResNet family are named ResNet-18, ResNet-34, etc. ResNet won the first place in the 2015 ImageNet competition, and the concept of identity skip connection is now widely used in almost all modern deep neural networks.

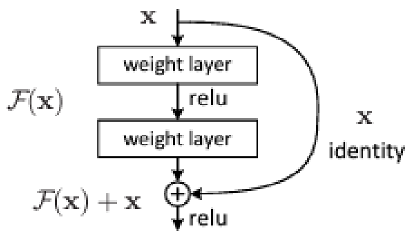

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 1: Residual Unit Proposed in (He et al. 2015)

Overview of the Project

In this mini-project, we are tasked with coming up with a modified residual network (ResNet) architecture, and train from scratch on the CIFAR-10 data set, aiming for the highest test accuracy. The model has a constraint of no more than 5 million parameters.

Methodology

Based on the implementation of ResNet from the GitHub repository https://github.com/kuangliu/pytorch-cifar, we tested out different variations on model architecture, data augmentation and selection of optimizers for the best test accuracy.

Model Architecture

Baseline Model: We use ResNet-18 as the baseline model, which passes data through an input stage, 4 stages of Residual Layers, and an output stage. ResNet-18 starts with an input layer, followed by the first convolutional layer with a kernel size of 3×3 and 64 input channels. Following the initial input stage are a sequence of residual blocks of 4 stages. Each residual block consists of two 3×3 convolutional layers followed by a batch normalization which normalizes the activations from the previous layer at each batch and a ReLU activation function to introduce non-linearity. Moreover, each residual block has a skip connection that bypasses one or more layers to avoid performance degradation. Residual blocks of same channel size are repeated a number of times in each stage before passed to the next, more-channeled stage. In the output stage, an average pooling layer reduces the spatial dimensions of the feature maps. Finally, a fully connected layer maps the pooled features to the class scores, which are then transformed into probabilities by a softmax activation function.

Modifications and Parameter Count: ResNet-18 has 18 learnable layers, with the number of channels increasing through 64, 128, 256, and finally 512 at various stages of the network. This architecture contains over 11 million trainable parameters. To satisfy the constraint of having no more than 5 million trainable parameters, the following architectural modifications have been implemented:

- Reduce the number of channels of each stage to 32, 64, 128 and 256 respectively, and change the repetition times of the residual blocks so that the total parameter is as close to 5 million as possible. With this, we have two variations with the repetition times of [8,6,4,3] and [3,3,5,3] for each stage. This modified ResNet architecture is illustrated in Figure 2.

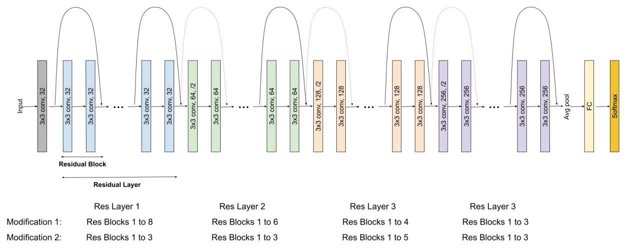


Figure 2: Modified ResNet Block Diagram 1

- Keep the number of channels same as the original ResNet-18 but reduce the repetition times of the residual blocks. With this, we have the repetition times to be [2,1,1,1] for each stage. This modified ResNet architecture is illustrated in Figure 3.

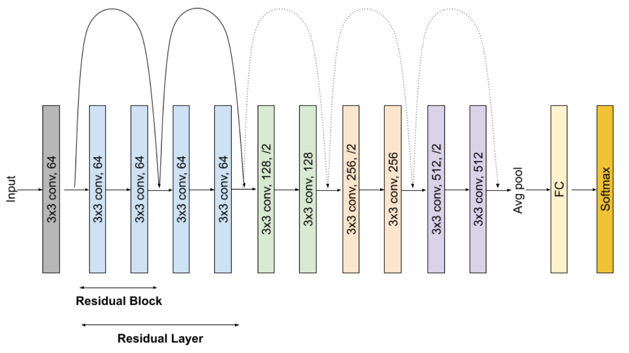


Figure 3: Modified ResNet Block Diagram 2

Pre-Activation: The structure of residual blocks with "pre-activation" was proposed in (He et al. 2016). The difference between the pre-activated and the original blocks can be seen in Figure 4. By moving the ReLU layer into the residual branch, information can be propagated directly from one unit to any other units, contrary to the original design that only positive result out of the addition can pass through the ReLU layer.

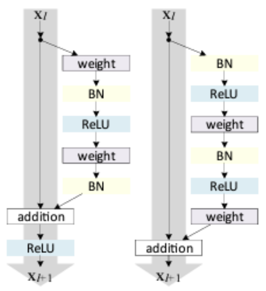


Figure 4: Left: original residual block in (He et al. 2015); Right: pre-activated block

Shortcut: In the original ResNet design, when the dimensions of input of a residual block $\mathbf{x}$ does not match output dimensions of $\mathcal{F}(\mathbf{x})$, the default projection shortcut is a 1×1 convolutional layer with stride of 2, which is throwing away a lot of information. Therefore, this paper (Duta et al. 2020) proposed to use instead a 3×3 max pooling with stride 2 followed by a 1×1 convolutional layer with stride of 1, to avoid waste of useful information. Figure 5 illustrates this change.

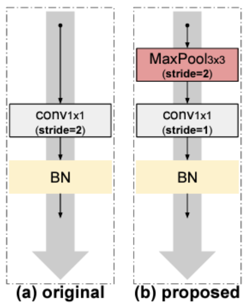


Figure 5: Proposed change of shortcut in (Duta et al. 2020)

Input Convolution: The original ResNet was trained on ImageNet, which feeds 224×224 sized data to the deep neural network. CIFAR's 32×32 images are significantly smaller, and each pixel might be more significant. So at the very first stage, instead of letting it pass through a 3×3 convolutional layer which is basically an average with surrounding pixels, we also experimented changing it to a 1×1 layer to let each pixel's information flow through easier.

Data Preprocessing

Dataset Overview: The CIFAR-10 dataset (Canadian Institute For Advanced Research) contains 60,000 32x32 color images in 10 different classes, with 6,000 images of each class. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The CIFAR-10 dataset is commonly used to train machine learning and computer vision algorithms.

Data Augmentation The following data augmentation techniques were experimented to encourage the generalization of the modified ResNet model.

- Random Crop: After padding the original image by 4 pixels, crop the image to a size of 32×32 pixels.
- Random Horizontal Flip: Horizontal flip the image randomly with a default probability of 0.5.
- Random Perspective: Applies a random perspective transformation with a specified distortion scale 0.3.
- Color Jitter: Randomly change the brightness within [0.5, 1.5], and hue within [-0.3, 1.3].

Training Process

Loss Function and Optimizer Cross-entropy measures the difference between the predicted probability distribution and the true distribution. It is chosen as the loss function in this task since it is commonly used for classification tasks such as CIFAR-10. As for the optimizer, we experimented with Stochastic Gradient Descent (SGD) with a learning rate of 0.1, a momentum of 0.9, and weight decay of 5×10^{-4}, and Adaptive Moment Estimation (Adam) with a learning rate of 0.001.

Hyperparameter Tuning Hyperparameters are variables that are manually set before training a model. To avoid the model being underfitting and overfitting, hyperparameter tuning is a necessary step. We experimented with different sets of hyperparameters by trial and error. More specifically, the hyperparameters in our experiments are batch size, learning rate, momentum, weight decay, and number of epochs. The effects of different sets of hyperparameters on training and testing accuracy will be presented in the result section.

Learning Rate Scheduler Learning rate schedulers seek to adjust the learning rate during training by reducing the learning rate according to a predefined schedule (Lau 2017). This adjustment helps to manage how fast a model learns. If the model learns too fast, it might not be able to capture complex relationships. If the model learns too slowly, it might get stuck in local minima. We used pytorch's cosine annealing learning rate scheduler, which adjusts the learning rate by a cosine function.

Result

30-epoch Comparison Tests

To explore the effects that different factors play on the result, we ran a series of 30-epoch comparison tests. In each of the tests, we varied one factor in question and fixed the others. Unless the factor is being tested or stated otherwise, the default combination of model and hyperparameters in this 30-epoch test is to use model structure [32,(8,6,4,3)] (please refer to subsection "Modifications and Parameter Count" for details), pre-activated and shortcut modified residual blocks, no convolution (equivalent to 1×1 Conv when implemented) in the input layer, batch size of 128, the first two data augmentation methods described above in "Data Augmentation", and the SGD optimizer with learning rate 0.1, weight decay 5×10^{-4} (no momentum).

Test1: This test was performed to see how the changes on block and net architectures would affect the result (test set accuracy). Parameter numbers for each case are also specified.

Table 1: Test1 result

Block	Input Layer w/ 3×3 Conv		No Conv	
	Original Shortcut	Modified Shortcut	Original Shortcut	Modified Shortcut
Basic	87.12% (4,976,426)	88.76% (4,976,246)	87.63% (4,975,658)	88.96% (4,975,658)
PreAct	87.95% (4,975,978)	88.96% (4,975,978)	88.00% (4,975,210)	89.71% (4,975,210)

Test2: This test focused on the sizes and residual block repetition times of the residual layers. Parameter numbers for each case are also specified.

Table 2: Test2 result

[32,(8,6,4,3)]	[32,(3,3,5,3)]	[64,(2,1,1,1)]
89.71% (4,975,210)	88.46% (4,955,882)	87.14% (4,974,794)

Test3: This test experimented two SGD optimizers, one without momentum (the vanilla SGD) and one with a momentum of 0.9, both with learning rate 0.1 and weight decay 5×10^{-4}. An Adam optimizer with a learning rate of 0.001 was also examined.

Table 3: Test3 result

SGD, Momentum=0.9	Vanilla SGD	Adam
83.84%	89.71%	90.6%

Test4: This test compared the result trained on two different data augmentations: only with Random Crop and Random Horizontal Flip, or these two combined with Random Perspective and Color Jitter. Experimented done with both vanilla SGD and Adam.

Table 4: Test4 result

	Random Crop, Random Horizontal Flip	Random Crop, Random Horizontal Flip, Random Perspective, Color Jitter
Vanilla SGD	89.82%	89.26%
Adam	90.55%	90.17%

Test5: This test compared the batch sizes of 64, 128, and 256. Experimented done with both SGDs.

Table 5: Test5 result

	64	128	256
Vanilla SGD	90.11%	89.71%	87.77%
SGD with Momentum	81.21%	84.86%	86.7%

Findings

After observing the test accuracy obtained in the 30-epoch comparison tests, we have observed the following:

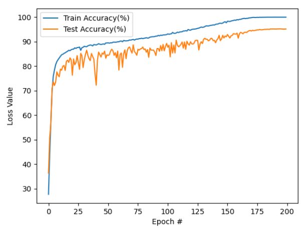
1. Using the Pre-Activation Block achieves higher testing accuracy than the Basic Block.
2. Using the Modified Shortcut achieves higher testing accuracy than the original shortcut.
3. Input layer without convolution achieves higher testing accuracy than the input layer with 3×3 convolution.
4. Using the model structure [32,(8,6,4,3)] achieves higher testing accuracy than the other two model structures.
5. Using the Adam optimizer achieves higher testing accuracy than the SGD optimizer with or without momentum.
6. Using only the Random Crop and Random Horizontal Flip data augmentation achieves higher testing accuracy than adding the additional Random Perspective and Color Jitter.

200-epoch Final Test Accuracy

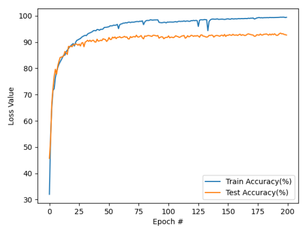
Based on the findings obtained from the 30-epoch comparisons, we ran a 200-epoch test using the following configuration and achieved a test accuracy of 95.25%:

- Model Structure: [32,(8,6,4,3)], Input Layer Without Convolution, Pre-Activation Block, Modified Shortcut.
- Data Augmentation: Random Crop and Random Horizontal Flip.
- Optimizer: SGD with a learning rate of 0.1, momentum of 0.9, weight decay of 5×10^{-4}.
- Batch Size: 128 for training and 100 for testing.

Figure 6a shows the train/test accuracy result over the 200 epochs for this final test.



(a) SGD 200-epoch test result



(b) Adam 200-epoch test result

Figure 6: 200-epoch Final Test results

Discussion

Due to the limit of GPU resources and time, we swept over the hyperparameter space with a small epoch number to pick out prospective candidates and ran those with the 200-epoch test. However, we found that those combinations with the highest score in the 30-epoch test do not always yield the best performance in the longer run. For example, Adam performed the best in the 30-epoch optimizer test, but when it is tasked with training more epochs, SGD would do better. It is discussed in (Keskar and Socher 2017) that Adam converges faster but SGD has better generalization and is preferred in image classification tasks. Figure 6b shows the train/test accuracy of another 200-epoch test using the same model setup as the previous one, but with Adam as the optimizer. It can be seen that its test accuracy increases fast and smoothly, but saturated around 50th epoch.

Another interesting observation is that all the modifications on the model architecture work in a way to allow information flow more easily from input to output. These changes do improve the test result, but for SGD with momentum the effect is not shown until a long test is run. Compared to the other 2 optimizers, its training loss stayed at higher levels, slowly but consistently driving the test accuracy up and eventually beats the performances of the others.

Conclusion

This project demonstrated the capabilities in image classification tasks of ResNet even under trainable parameter constraints and the importance of choosing the appropriate model configuration, including model architecture, hyperparameters, data augmentation, and optimizer, in achieving optimal performance.

References

Duta, I. C.; Liu, L.; Zhu, F.; and Shao, L. 2020. Improved Residual Networks for Image and Video Recognition. *arXiv preprint arXiv:2004.04989*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Identity Mappings in Deep Residual Networks. In *European Conference on Computer Vision*.

Keskar, N. S.; and Socher, R. 2017. Improving Generalization Performance by Switching from Adam to SGD. *arXiv:1712.07628*.

Lau, S. 2017. Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning. https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1. Accessed: 2024-04-10.