Mini-Project Report

Hao Wang, Puda Zhao, Yinhong Qin

New York University
Deep Learning Fall 2022 ECE-GY 7123
hw2671@nyu.edu, pz2078@nyu.edu, yq2021@nyu.edu

Abstract

Deep neural network model can estimate more complicated function theoretically but it will suffer from the gradient explosion and vanishing as well as the exponentially growth of the number of parameters. Residual neural network (ResNet) can improve these problems and help researchers build very deep network. In this mini-project, we propose a image classification model based on ResNet model with the total number of parameters less than 5M. Our model was trained and tested on CIFAR-10 dataset and can reach 93.8% accuracy. We also made several variations of changes to select the final model with best testing performance on this dataset.

Overview

We trained a ResNet architecture to classify images from the CIFAR10 dataset with maximum around 0.94 accuracy.
We used several useful techniques in this project to improve our model performance and reduced the training time, i.e., Data normalization, Data augmentation, Learn-ing rate scheduling, etc. We will explain in detail in the sub-sequent section.
We tried several different Block Number combinations and found that the closer the change of Block Number is to the input, the smaller the impact on the number of pa-rameters, and the closer to the output, the greater the im-pact on the number of parameters.
We adjusted the parameters on the default ResNet-18 model, and achieved better accuracy than the default model with a more simplified model. The training time for 70 epochs is about 25 minutes.
Our codebase can be accessed throught this GitHub repository: https://github.com/KevinQyhinNYU/Deep-learning-Mini-project/tree/master
It is modified based some online examples: For code source part, we checked and did modifications based on: https://www.kaggle.com/code/datajameson/cifar-10-object-recognition-resnet-acc-94.
For ResNet design part, our model is based on: https://github.com/kuangliu/pytorch-cifar/blob/master/models/resnet.py.

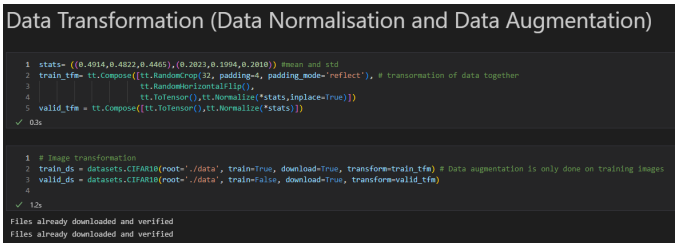

Figure 1: Data normalization and augmentation

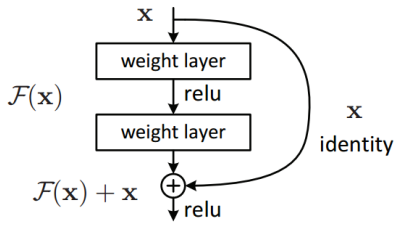

Figure 2: simple model of ResNet

Methodology

Data normalization: We normalized the image tensors by subtracting the mean and dividing by the standard deviation of pixels across each channel. Normalizing the data prevents the pixel values from any one channel from disproportionately affecting the losses and gradients.
Data augmentation: We applied random transformations while loading images from the training dataset. Specifically, we will pad each image by 4 pixels, and then take a random crop of size 32 x 32 pixels, and then flip the image horizontally with a 0.5 probability.
Residual connections: One of the key changes to our CNN model was the addition of the residual block, which adds the original input back to the output feature map obtained by passing the input through one or more convolutional layers.
To seamlessly use a GPU, if one is available, we define a couple of helper functions.
Batch normalization: After each convolutional layer, we added a batch normalization layer, which normalizes the outputs of the previous layer. This is somewhat similar to data normalization, except it's applied to the outputs of a

Figure 3: GPU functions

layer, and the mean and standard deviation are learned parameters.

Learning rate scheduling: Instead of using a fixed learn-ing rate, we will use a learning rate scheduler, which will change the learning rate after every batch of training. There are many strategies for varying the learning rate dur-ing training, and we used the "One Cycle Learning Rate Policy".

Weight Decay We added weight decay to the optimizer, yet another regularization technique which prevents the weights from becoming too large by adding an additional term to the loss function.

Gradient clipping:We also added gradient clipping, which helps limit the values of gradients to a small range to prevent undesirable changes in model parameters due to large gradient values during training.

Results

We designed a 4-layer ResNet with block numbers equal to [2, 1, 1, 1]. We tried different cases by setting different weight-decay and finally reached maximum 0.9381 accuracy with total 4977226 of parameters.

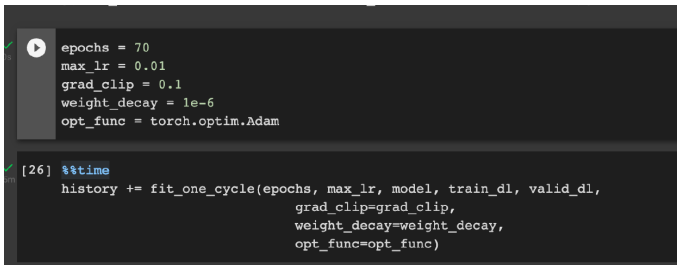

Figure 4: Settings

Figure 5 to 8 show the information about training process and testing results of this model. Based on the loss curve of the training and validation process, we can notice that this model does not has the overfitting issue on the CIFAR-10 dataset.

Figure 9 is a visualization result showing how the learning rate was changing over the training epochs. We can notice

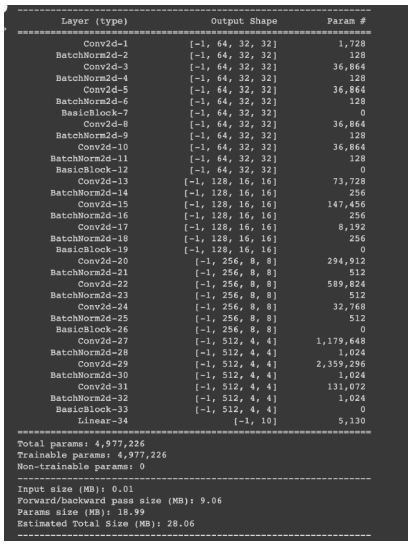

Figure 5: Parameters

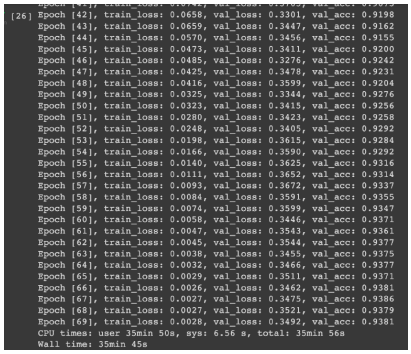

Figure 6: Parameters

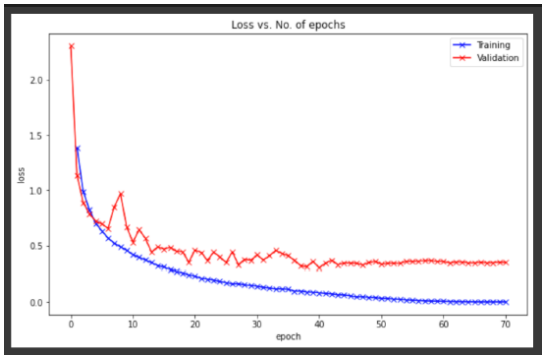

Figure 7: Loss vs No. of epochs

that the learning rate starts at a low value, and gradually increases for 0.3 of the iterations to a maximum value of 0.01, and then gradually decreases to a very small value.

Findings and other testing results

Cutting the number of channel for each residual layers, the number of trainable parameters drops dramatically. Adding

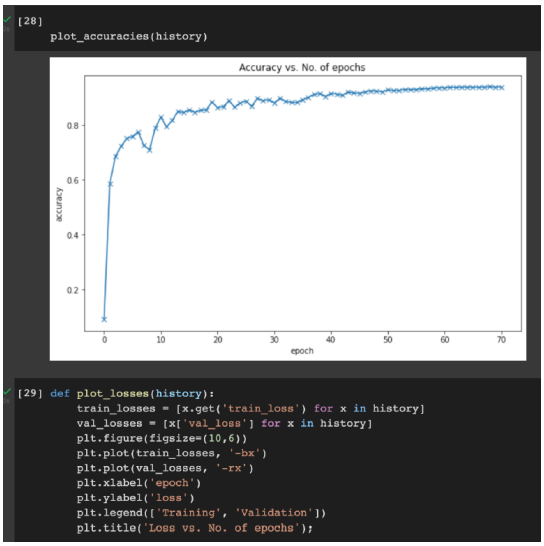

Figure 8: Accuracy vs No. of epochs

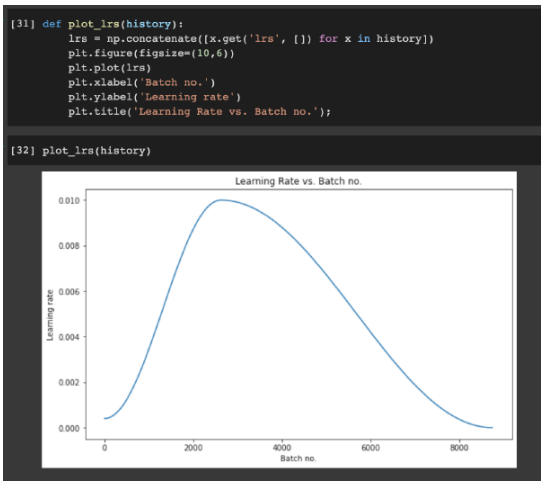

Figure 9: Learning Rates vs No. of epochs

more residual blocks in each layers the trainable parameters increases. For comparison, we keep two different model's trainable parameters around 4.9 million. The first model has original number of channel and one residual block in each layers. The second model cuts the number of channel in half for each layer and increases the residual block number to 6 for first layer, 5 for second layer, 4 for third layer and 3 for fourth layer. The best accuracy we get from first model is 0.9338 and the best accuracy we get from second model is 0.9387. Apparently the second model is slightly better than the first one. Therefore increasing the residual block number for each layer has better reward compared to increasing the channel numbers for each layer.
Our main findings are listed as follows:

- The default ResNet-18 model is excessive in CIFAR-10 classification. Simpler models are sufficient to satisfy this project goal. Our model could reach around an accuracy

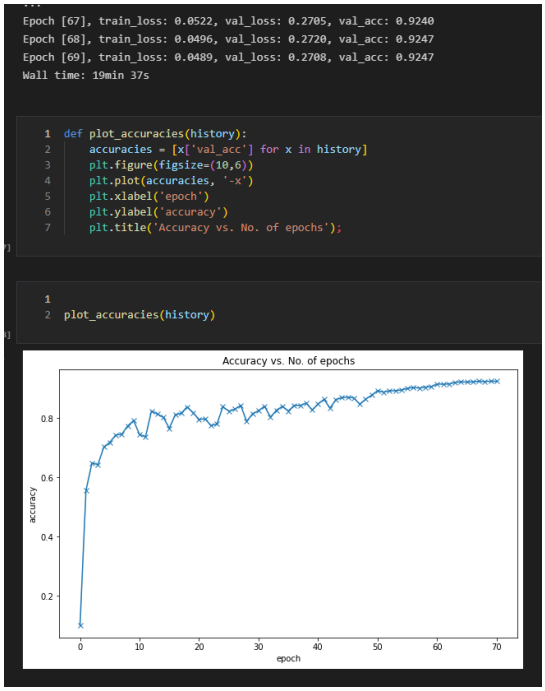

Figure 10: Accuracy vs No. of epochs

of almost accuracy 0.94 with far less that 5 million parameters.

- The width of ResNet design is more important than the depth. With holding the same parameter numbers, a deeper ResNet may work worse that a wider one. This will illustrates the idea of ResNet. As shown in the figure 10, a ResNet model with less than 1 million parameters could also reach more than 0.92 accuracy. It proved that it is more important to do modifications on the width of the model.
- In terms of data normalization and augmentation, it is useful. Normalization helps to get the data within a range (we specify) and which helps in making training a lot faster. Image augmentation is a super effective concept when we don't have enough data with us.
- In terms of hyper parameters, a too high learning rate leads to a suffering at first, large batch size makes little difference in our test.

References

He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.