EE569 Digital Image Processing

HOMEWORK #5 Competition

Issued: 03/23/2020
Problem2 due: 05/03/2020

Yao Fu
6786354176
yaof@usc.edu

EE 569 Homework #5
May 03, 2020

Problem 2: EE569 Competition --- CIFAR10 Classification (50%)

Find some papers about studying the CIFAR10's classification, the use the knowledge you acquire to do some modifications of the baseline CNN from the Problem 1 of Homework 5 with the purpose of enhancing the classification accuracy.

2.1 Abstract and Motivation

Deep learning, which is based on artificial neural networks, is one of the most popular machine learning methods. In the field of deep learning, convolutional neural networks (CNNs) , most widely applied to computer vision ranging from image recognition to texture classification, are one class of deep neural networks, whose name indicates that the architecture of networks utilizes a mathematical operation called convolution. CNNs are regularized versions of multilayer perceptrons, by making good use of the hierarchical pattern in data and assembling more complex patterns using smaller and simpler patterns.

In this assignment, I will modify the trained CNN from the Problem 1 in order to enhance the CIFAR-10 classification by using Keras/Python. Necessary results and images will be presented later in my report.

2.2 Motivation and logics behind my design

The interest in convolutional neural networks has been rooted in my mind for a long time. The VGG16, which is a novel and elegant convolutional neural network model famous for its achievement in ILSVRC-2014, is able to make test accuracy reach 92.7% top-5 in the ImageNet dataset. This impressive model is proposed by K. Simonyan and A. Zisserman from the paper[1]. Due to a fact that VGG16 is designed for the giant ImageNet datasets, it's highly probable for us to confront the overfitting issue when applying the VGG16 to the Small datasets like CIFAR-10.

However, I am really fond of the VGG16's legend and have discovered the paper[2], from which I have reached the point that the authors have done some modifications to enable the very deep VGG16 network, whose architecture is shown in the Fig. 2.1, to fit the small CIFAR-10 datasets without losing the VGG16's strong power. In order to achieve the high-test-accuracy goal, based on the authors' work, I will make several modifications of the traditional architecture of VGG16 shown in the Fig. 2.1.

First of all, I will shrink the two 4096-dimension fully connected layers to one 100-dimension fully connected layer, thus making it possible to reduce the number of parameters.

Secondly, I will use the batch normalization from[3] to normalize each layer's inputs to accelerate the network's convergence, which will also result in reducing error rate and training time simultaneously because Batch Normalization allows people to set learning rates much higher and be less careful about initialization of weights without considering its adverse influence on the test accuracy according to the paper [3]. Specifically speaking, I will add batch normalization layers before every nonlinear layer.

Last but not least, according to the paper[2], I am about to add more "dropout" parts (0.4 or 0.5) and set a stronger weight decay value (0.0005) to reduce overfitting. Instead of just putting the dropout at the fully connected layers where there are a great many parameters, I will also deploy dropout parts for convolutions layers.

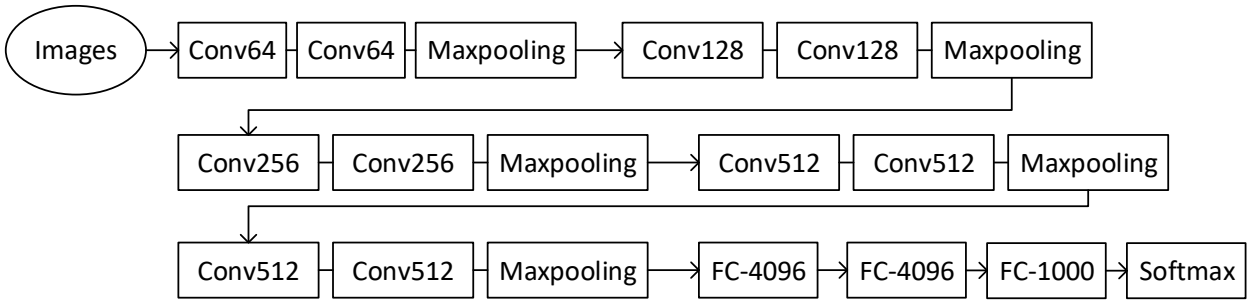

Fig. 2.1: The simple CNN architecture derived from VGG-net

In the meanwhile, while training my model common techniques conductive to improving test accuracy will also be considered such as normalizing the dataset to reduce training time and augmenting data to increase the diversity of data. As for other common parameters, I will set "batch size" as 128, "learning rate" as 0.1, and "learning-rate decay" as 1e-6.

In a word, compared with the simple model in my Problem 1 of Homework 5, the method proposed by the paper[2] can tremendously increase the accuracy of CIFAR-10 classification with power of a very deep convolutional neural network. This can also make the model have high speed of convergence and low degree of overfitting by utilizing the batch normalization and dropout setting. However, compared with the method of Problem1, the overfitting issue of the new method is still severe, and the training time is significantly long.

2.3 Classification accuracy

Due to the complete first time for me to do deep learning projects by using AWS, I have made a great many silly mistakes such as running 100 epochs with an expensive instance type without checking my program's validity carefully, thus resulting in my high billings as shown in the Fig 2.2, which also includes some cost while implementing Successive Subspace Learning of Homework 6 before figuring out how to use batch processing to save memory. Even if I am confronted with a high cost after accomplishing this assignment, yet I am still satisfied since I have acquired a big amount of fundamental and practical knowledge of convolutional neural networks and learned some valuable and unforgotten lessons when wanting to use AWS to run programs, which, I believe, will play a significant role in my future research on deep learning. The summary of my classification accuracy in this competition is presented below.

My best test accuracy is 0.8372 as shown in the Fig. 2.3. But honestly speaking, due to my carelessness, my improper preprocessing data made me have test accuracy 0.1 at first, which I ran several times with an expensive instance type to figure out that mistake. Once adjusting my

code correctly, I just made a trial to run 50 epochs and then I get a test accuracy of 0.8372 as shown in the Fig. 2.3 without saving the test and loss curves. Later, I ran another 50 epochs and the test accuracy is only 0.8034 which made me really disappointed. Finally, I understood that my number of epochs is not big enough to make the train and test accuracy converge, which exactly explained the phenomenon that I get two different values of test accuracy with the same number of epochs largely because of the fluctuations of test curves. The I found that the original model by the authors from [2] has to be trained over 200 epochs, which indicates that my modified model should at least be trained 100-200 epochs instead of merely 40 or 50. However, due to my disappointing bills shown in the Fig. 2.2, I don't want to train my model again for 100 epochs although it may make the test accuracy up to 0.9 theoretically. Therefore, eventually I decide to just let the value of 0.8372 be my final result in this assignment.

As I have mentioned before, I forgot to save the training time and performance curves for the case when the test accuracy is 0.8372 because I wasn't familiar with "WinSCP" at that time. The performance curves shown in the Fig. 2.8 and the Fig. 2.9 are the results when the number of training samples is 50k, but the number of epochs is 40 which is another trial. The sudden waves of accuracy curves can make sense that the value of test accuracy will have some fluctuations within 40 or 50 epochs. The sum of training and inference time is 55989.27 seconds while the inference time is only around 6 seconds with the t2.2xlarge instance type as shown in the Table 1.

Moreover, I use the built-in function *train_test_split* to randomly drop training samples. The results are shown in the Table 1. The Fig. 2.4 and the Fig. 2.5 are the performance curves when the number of training samples is 10k, and the Fig. 2.6 and the Fig. 2.7 are the performance curves when the number of training samples is 30k. From the Table 1 and the Fig 2.10, we could draw a conclusion that the reduction of training samples does degrade the test accuracy but not too much compared with the traditional Lenet5, indicating that my model is more powerful.

Table 1: The experiment results with different number of the train samples

Number of Train Samples	5k	10k	30k	50k
Processor	Intel-i7- 10	Intel-i7- 10	Intel-i7-10	t2.2xlarge
Time	11443.25s	22986.12s	60857.55s	55989.27s
Test Accuracy	0.4622	0.5417	0.7578	0.7924
Epoch	50	50	50	40

Fig. 2.2: My bills after finishing homework5&homework6

```
The loss for the testing samples: 1.467196351967574
The accuracy for the testing samples: 0.8371666519972341
```

Fig. 2.3: The screen shoot of my best test accuracy within my experiments

Fig. 2.4: The accuracy curves' plot on training and testing datasets when the number of train samples is 10k

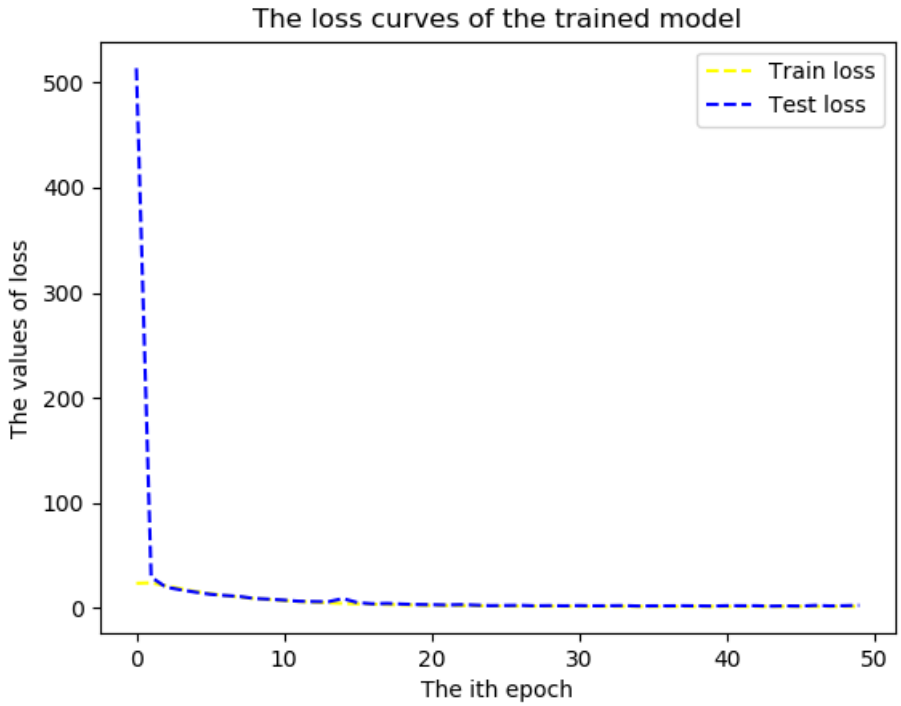

Fig. 2.5: The loss curves' plot on training and testing datasets when the number of train samples is 10k

Fig. 2.6: The accuracy curves' plot on training and testing datasets when the number of train samples is 30k

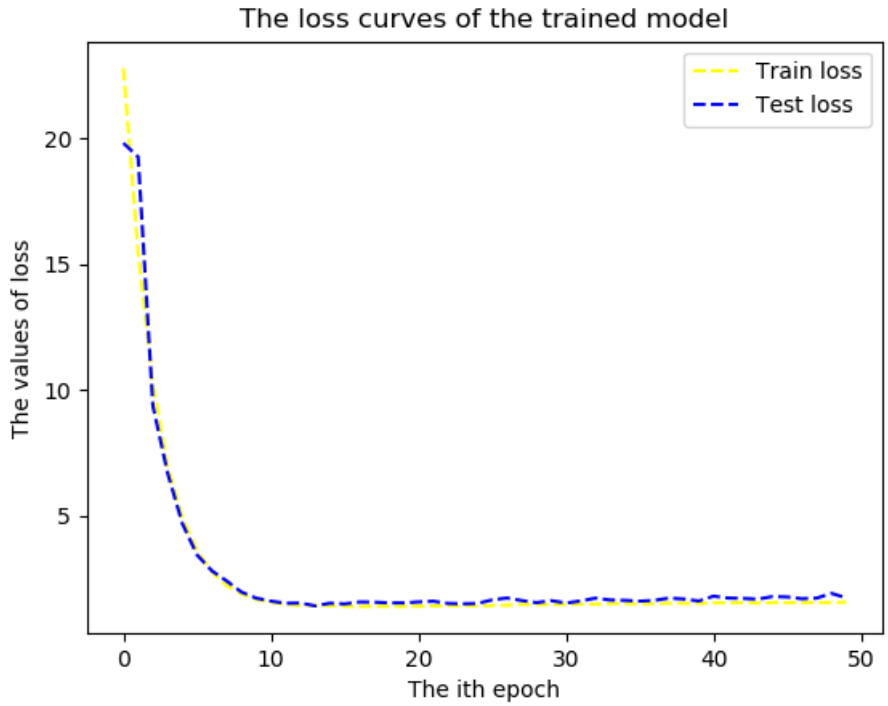

Fig. 2.7: The loss curves' plot on training and testing datasets when the number of train samples is 30k

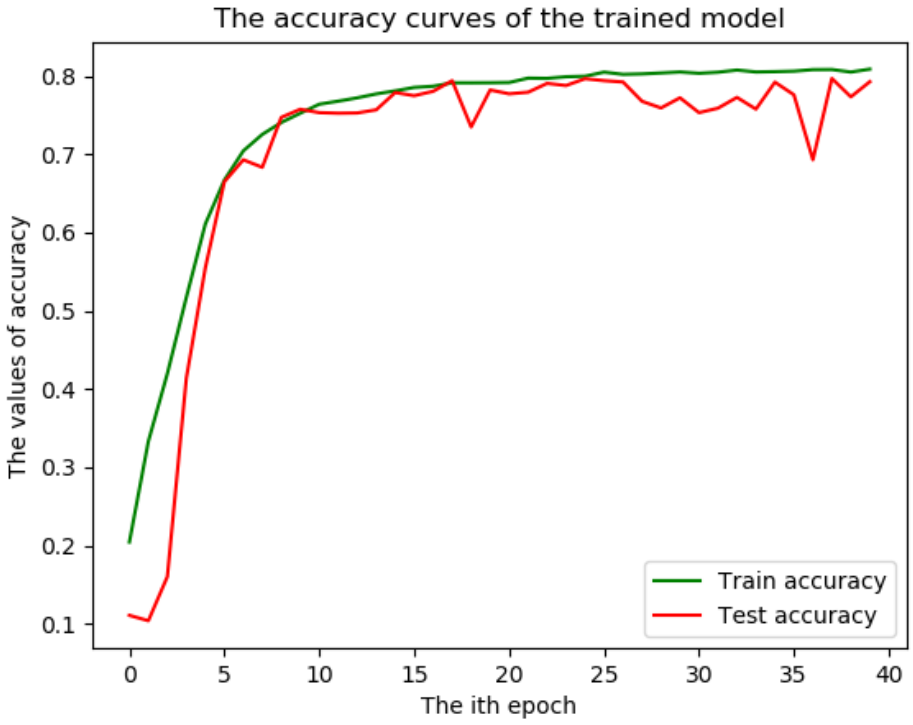

Fig. 2.8: The accuracy curves' plot on training and testing datasets when the number of train samples is 50k

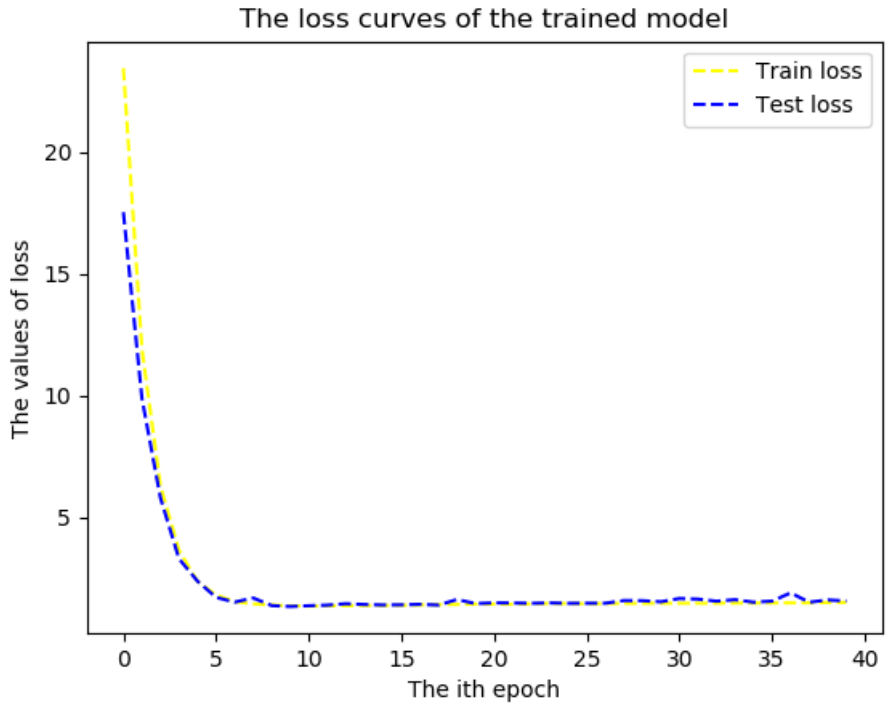

Fig. 2.9: The loss curves' plot on training and testing datasets when the number of train samples is 50k

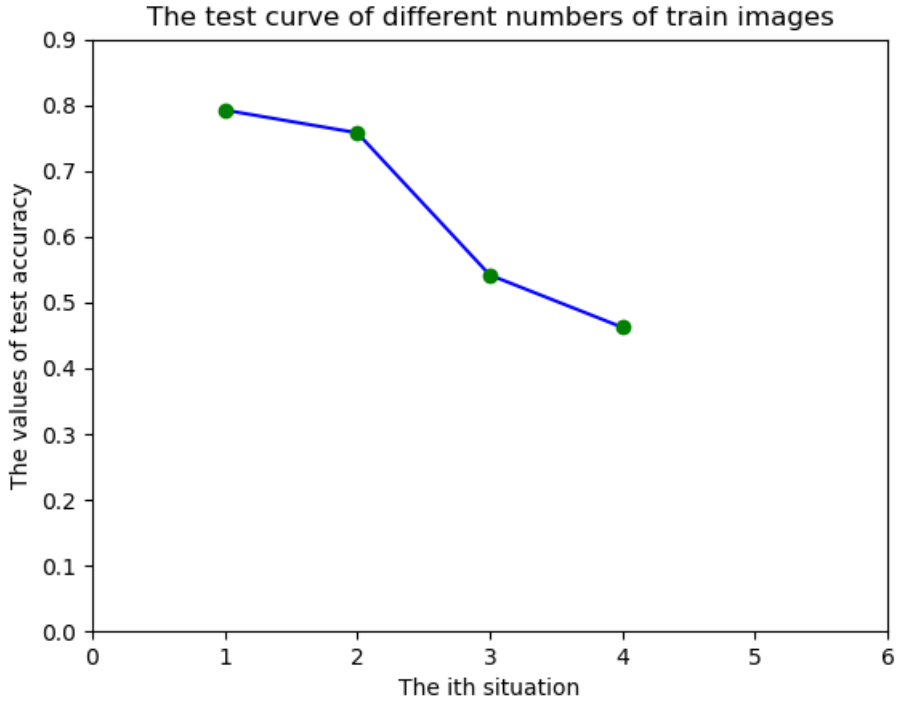

Fig. 2.10: The test curve of different numbers of train images from the Table 1, where the x axis 1, 2, 3, and 4 represent 50k, 30k, 10k, and 5k train images

2.4 Model size

The size of my trained is shown below:

Total params: 15,001,418
Trainable params: 14,991,946
Non-trainable params: 9,472

Compared with my friends' model, the number of my parameters is very large, which, I believe, can explain why I am supposed to apply more epochs to make performance curves converge. However, due to some financial reasons I didn't try increasing the number of epochs to 100 or more.

2.5 Summary

In this competition, my model's test accuracy is 0.8372 and size is 15,001,418.

References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
[2] Shuying Liu and Weihong Deng. Very deep convolutional neural network-based image classifi- cation using small training sample size. In Pattern Recognition (ACPR), 2015 3rd IAPR Asian Conference on, pages 730–734. IEEE, 2015.
[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.