EE569 Digital Image Processing

HOMEWORK #5

Issued: 03/23/2020
Problem1 due: 04/07/2020
Problem2 due: 05/03/2020

Yao Fu
6786354176
yaof@usc.edu

EE 569 Homework #5
April 07, 2020

Problem 1: CNN Training on LeNet-5 (100%)

(a) CNN Architecture (Basic: 20%)

(b) CIFAR-10 Classification (Basic: 50%)

(c) State-of-the-Art CIFAR-10 Classification (Advanced: 30%)

1.1 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 order to understand the basic mechanism of convolutional neural networks, in this report, I will train a simple CNN called the LeNet-5 from [1], which is designed for handwritten or digitally printed character recognition, and apply it to the CIFAR-10 dataset [2] by using Keras/Python3 to achieve them. Necessary results and images will be presented later in my report.

1.2 Approach and Procedures

(a) CNN Architecture

Question (1):

i. The fully connected layer

In the architecture of neural networks, there are always a certain number of neurons, which can be also regarded as activation units, in each single layer. Fully connectedness means that every neuron belonging to one layer must be connected to all the neurons in another layer, which has exactly the same meaning as the multi-layer perceptron neural network (MLP).

ii. The convolutional layer

The term “convolutional” comes from the field of digital signal processing. However, according to the professor’s clarification in class, the essence is “correlation” but individuals in

academia still prefer to call “convolution”, which I have understood that tricky ambiguity. In the architecture of CNN, convolutional layers play an indispensable role in extracting features, which is realized by convolving input data or signal with different types filters (dot production) and transfer results (activation maps) to the next layer. From my point of view, this is the same idea or concept that we use a Gaussian filter to convolve the image if we want to remove noise.

Although fully connected neural networks can be theoretically utilized to attain features from image data containing large sizes, it is not practical in real life due to limitations of computer hardware. Fortunately, the emergence of convolution operation makes it possible to largely reduce the number of free or unimportant parameters of input data, thus highly lowering the burden of computers.

iii. The max pooling layer

Pooling layers are served as reducing dimensions or spatial sizes of representation by transforming several neurons within one patch size (usually 2 by 2 or 3 by 3) at one layer into one neuron in the next layer, which can also be regarded as a process of nonlinear down-sampling. There are usually two approaches to achieving pooling: max pooling one and average pooling.

Empirically, people are likely to choose max pooling, which indicates using the maximum value among neurons located in a patch (typically 2 by 2) as the output result to discard 75% of the activations, because max pooling can frequently achieve more desirable results than average pooling.

iv. The activation function

In the field of neural networks, one node’s activation function defines its output result of given one input or several inputs. This concept reminds me of a simple digital circuit that can be "True" (1) or "False" (0), depending on the input signal.

There are plenty of functions that can perform activation, among which ReLU, the abbreviation of rectified linear unit, is widely used in the neural networks. The mechanism of ReLU can be summed up as $f(x) = \max(0, x)$, which totally removes negative values from activation maps by replacing them with zeros. It’s conductive to increasing the nonlinear properties of the overall network without influencing receptive fields of convolution layers.

v. The softmax function

The "loss layer" specifically shows how much error between predicted results and true labels, which is commonly the final layer of a neural network. There are a variety of loss functions, such as softmax loss, sigmoid cross-entropy, and Euclidean loss, that are adequate for different tasks.

In mathematics, the softmax function is referred to a function that can make K-real-number input vector normalized into a probability distribution, which are composed of K probabilities proportional to the input numbers’ exponentials. As a result, no matter what initial situations are, after applying softmax, each component will be in the interval $(0,1)$, and their sum will be equal to 1, thus leading to being interpreted as probabilities. The standard (unit) softmax function $\sigma: R^K \to R^K$ is defined by the formula

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \; for\; i = 1, \dots, K \; and \; z = (z_1, \dots, z_K) \in R^K.$$

The softmax function is always a good choice for the final layer of a neural network when a task requires us to predict a single class of K mutually exclusive classes.

Question (2):

i. The over-fitting issue

While utilizing machine learning algorithms with the help of a certain amount of training data to train a statistical model, that resulting model excessively seeks for low errors from the training data set without taking its predictive power into account. As a result, the overfitting model will poorly perform predictions, thus leading to higher errors by comparison with testing data. In other words, this awful model, in essence, has merely finished "memorizing" training data rather than "learning" to build a desirable model. For example, we want to train a model that should have learnt the art style of one image from Da Vinci. Unfortunately, the overfitting model will not only learn the art style, but also learn the noise from that image, thus resulting in poor visual quality.

ii. Techniques to avoid over-fitting in CNN

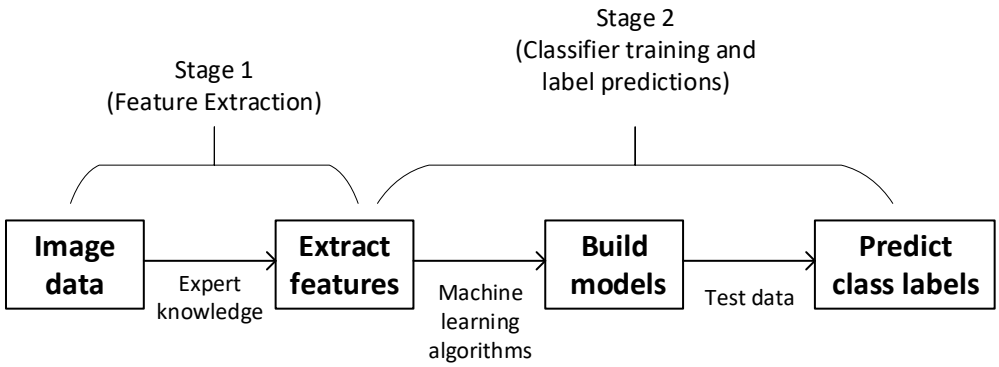
Generally speaking, the CNN apply the regularization methods to prevent overfitting. CNNs use various types of regularization. When it comes to how to regularize it, there are usually two types: empirical and explicit.

For the empirical way, first of all, we could discard some individual nodes with a certain probability p to reduce dimensions of networks because a fully connected layer contains all the weighting coefficients, which can lead to overfitting easily. Secondly, we can also think about random pooling, which means that we pick the activation within each pooling area randomly in terms a multinomial distribution. Thirdly, we could create some artificial data as training samples such as getting purely new data or change existing data slightly as new ones.

For the explicit way, first of all, we could simply stop training early before overfitting is likely to occur. It comes with the disadvantage that the learning process is halted. Secondly, we could limit the number of parameters, ranging from limiting the number of hidden units to using max pooling to reduce dimensions. Thirdly, we could simply regard the weight decay as the added regularizer. Lastly, an absolute upper bound can be introduced to constrain the magnitude of the weight coefficients of all the neurons.

Question (3):

The traditional methods to resolve image classification problems can be shown as follows:



From the flowchart, it's not hard for to know that the traditional methods can only focus on the "stage 2" to enhance predicting performance while the "stage 1" requires expert knowledge that usually individuals studying machine learning algorithms don't have. In that case, the features are the king. However, with the magnificent emergence of convolutional neural networks, the original pure data can become the king because we don't demand domain knowledge to extract features anymore. With the strong assistance of the CNN, we are capable of directly handling the original data and finally getting the cost function. Later, all we need to do is to guarantee that the cost function can be minimized by the gradient descend method to make networks achieve good performance without trying to acquire good features.

Question (4):

i. The loss function

Theoretically speaking, a loss function can be regarded as a function mapping one or more variables onto a real number, which can represent the "cost" of our study objective that we can utilize optimization techniques to minimize the loss. In the architecture of the CNN, the last layer is always the "loss layer" that specifies the deviation between the predicted outputs and true label values. There are a number of loss functions that can be used. For example, we can utilize the softmax function to find out softmax loss and then apply the backpropagation method, which will be discussed later, to optimize it by adjusting different types of parameters, making it possible to enable neural networks to achieve the best predicting performance.

ii. The backpropagation

In order to make a trained neural network have good predicting ability, it's not an unwise idea that we choose backpropagation to minimize the loss function. In detail, during that process there is an algorithm calculating the gradient of the loss function in terms of the weighting coefficients of the network by the chain rule and iterating backward from the last layer to get rid of superabundant calculations of intermediate terms efficiently, which makes it feasible to utilize gradient methods to update all types of parameters to finally minimize loss.

(b) CIFAR-10 Classification

In this part, I will use Keras based on TensorFlow to train a simple convolutional neural network architecture based on LeNet-5 as shown in Fig. 1.1.

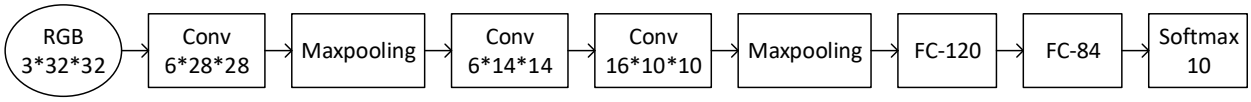


Fig. 1.1: The simple CNN architecture derived from LeNet-5

I will use the built-in function "keras.utils.to_categorical" to preprocess the input image data. After getting the training history, I will use the Python's packetage "matplotlib" to plot performance under 5 different initial parameter settings, where initialization methods can be "glorot_uniform", "glorot_normal", and "he_uniform", different values of learning rate and decay will be also tried. Finally, the best performance curve within my experiments will be given. The performance curves and detailed discussion will be shown later.

(c) State-of-the-Art CIFAR-10 Classification

In this part, I choose the paper[3] for the discussion about the state-of-art implementation on CIFAR-10 classification.

VGG16 is a novel and elegant convolutional neural network model, which is proposed by K. Simonyan and A. Zisserman from the paper[4]. The model, famous for its achievement in ILSVRC-2014, can make test accuracy reach 92.7% top-5 in the ImageNet dataset. Hence, it's not hard for us to notice that VGG16 is trained on the giant ImageNet datasets, thus leading to overfitting for the Small datasets like CIFAR-10.

However, from the paper[3], I have reached the point that the authors have done some modifications to enable the very deep VGG16 network to fit the small CIFAR-10 datasets without losing the VGG16's strong power, whose architecture is shown in Fig. 1.2. In order to achieve the goal, the authors will make several following modifications of the Fig. 1.2.

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

Secondly, the authors use the batch normalization from[5] to normalize each layer's inputs to accelerate the network's convergence, thus resulting in reducing error rate and overfitting degree. Specifically speaking, the authors will add batch normalization layers before every nonlinear layer.

Last but not least, the authors add more "dropout" parts and set stronger weight decay values to reduce overfitting. Instead of just putting the dropout at the fully connected layers where there are a great many parameters, the authors will also set dropout for convolutions layers.

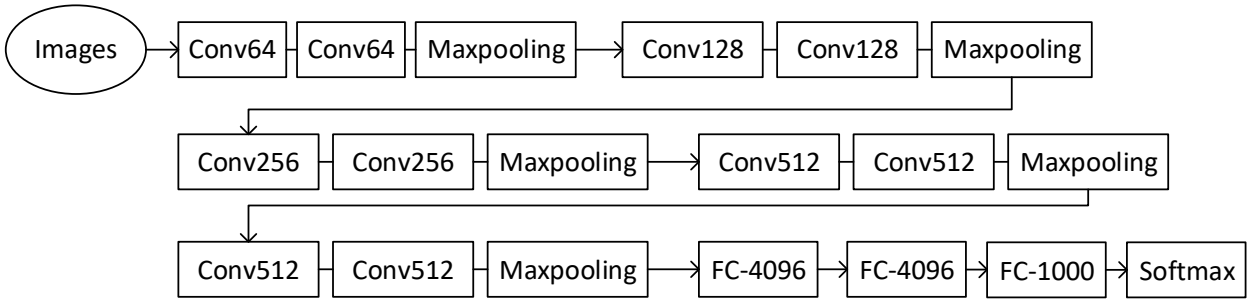


Fig. 1.2: The simple CNN architecture derived from VGG-net

To sum up, compared with the simple method-A shown in Fig. 1.1, the method-B proposed by the paper[3] can tremendously increase the accuracy of CIFAR-10 classification with power of a very deep convolutional neural network. The method B 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 A, the overfitting issue of the method B is still severe, and the training time of the method B is significantly long.

1.3 Experimental Results

Table 1: The accuracy results of the test set in different situations

	Fig. 1.3	Fig. 1.4	Fig. 1.5	Fig. 1.6	Fig. 1.7	Fig. 1.8	Fig. 1.9
Accuracy	0.5872	0.6021	0.6046	0.6068	0.6030	0.6117	0.5620

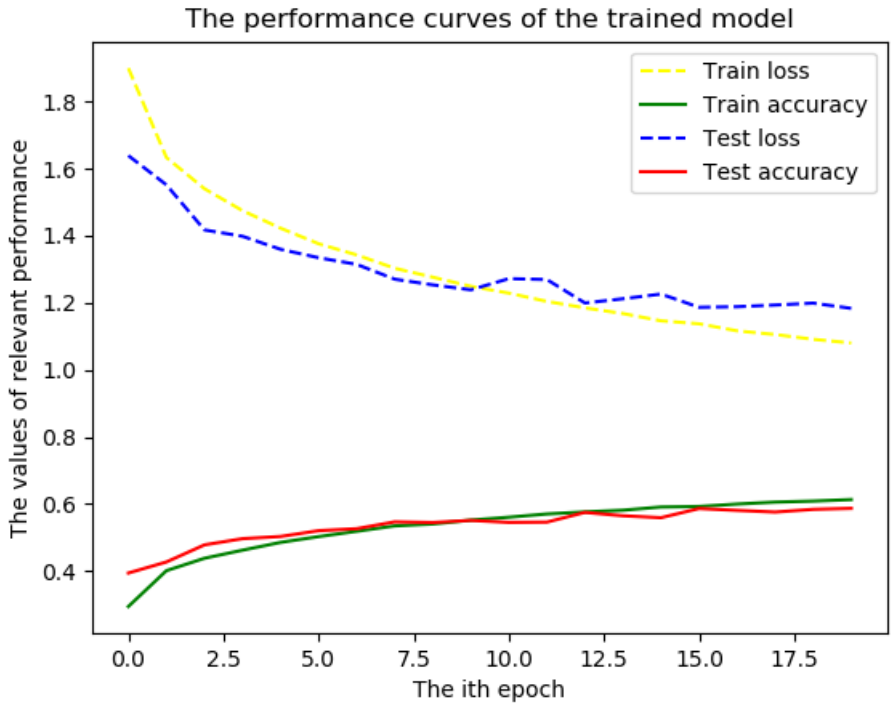


Fig. 1.3: The performance curves' plot on training and testing datasets
when the initializer is "glorot_uniform", the learning rate is 0.02, the decay is 0.0001

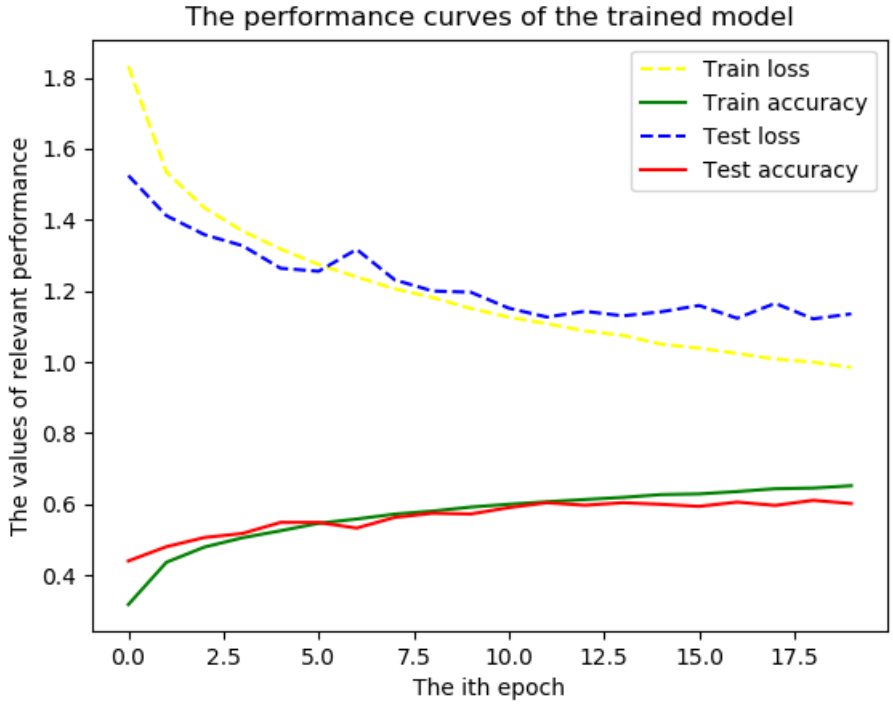


Fig. 1.4: The performance curves' plot on training and testing datasets
when the initializer is "glorot_uniform", the learning rate is 0.01, the decay is 0.00001

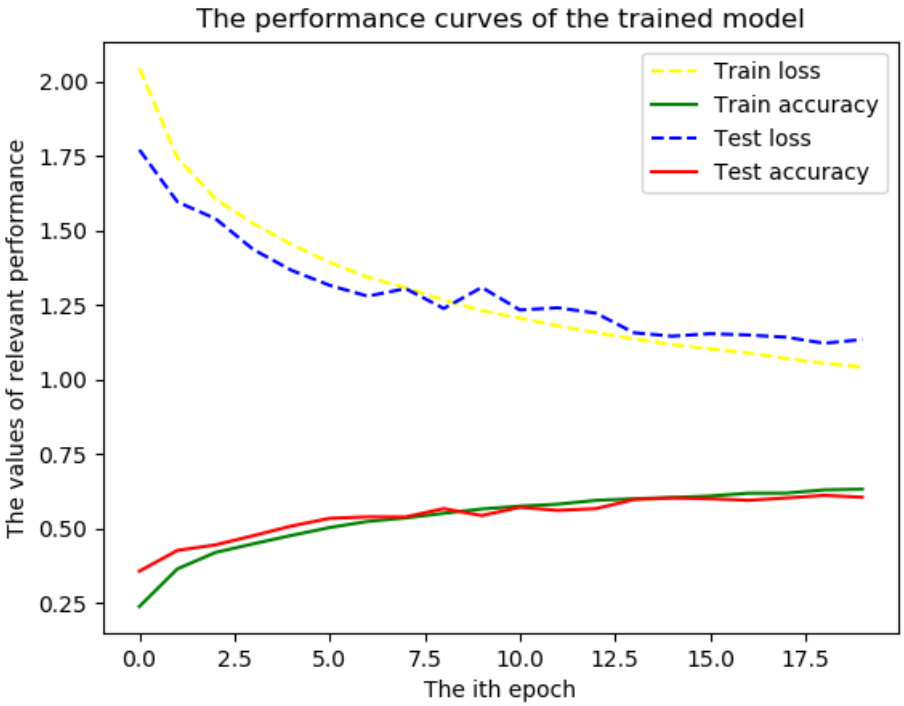


Fig. 1.5: The performance curves' plot on training and testing datasets
when the initializer is "glorot_normal", the learning rate is 0.03, the decay is 0.0002

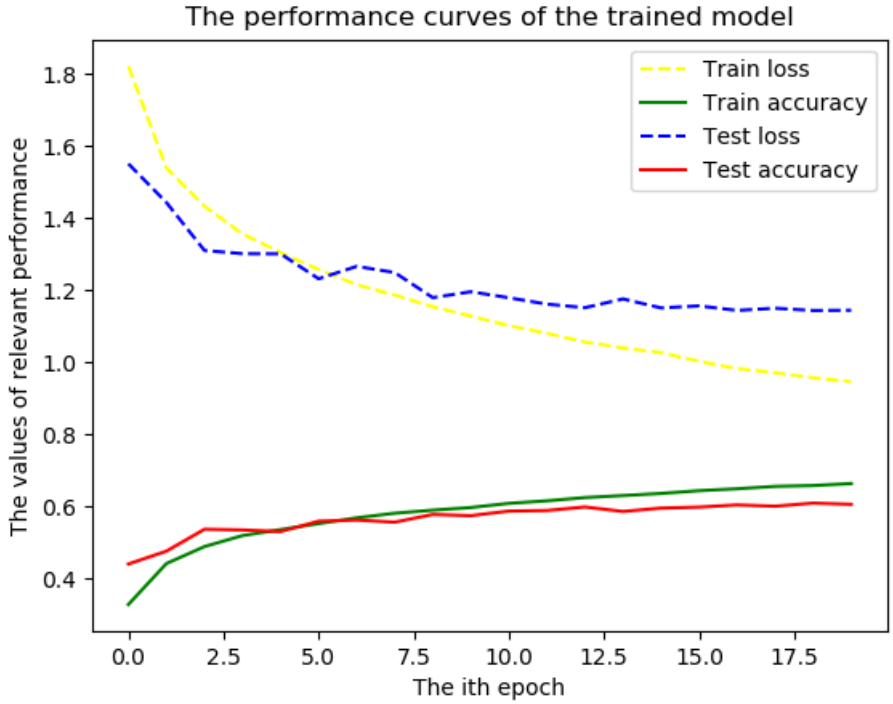


Fig. 1.6: The performance curves' plot on training and testing datasets
when the initializer is "he_uniform", the learning rate is 0.025, the decay is 0.0001

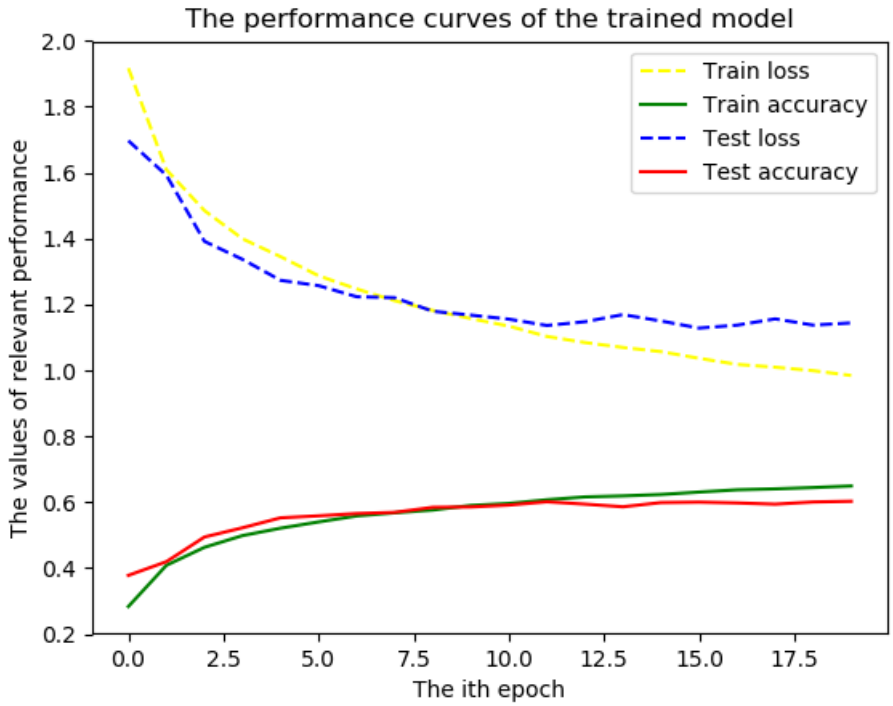


Fig. 1.7: The performance curves' plot on training and testing datasets
when the initializer is "glorot_uniform", the learning rate is 0.04, the decay is 0.0003

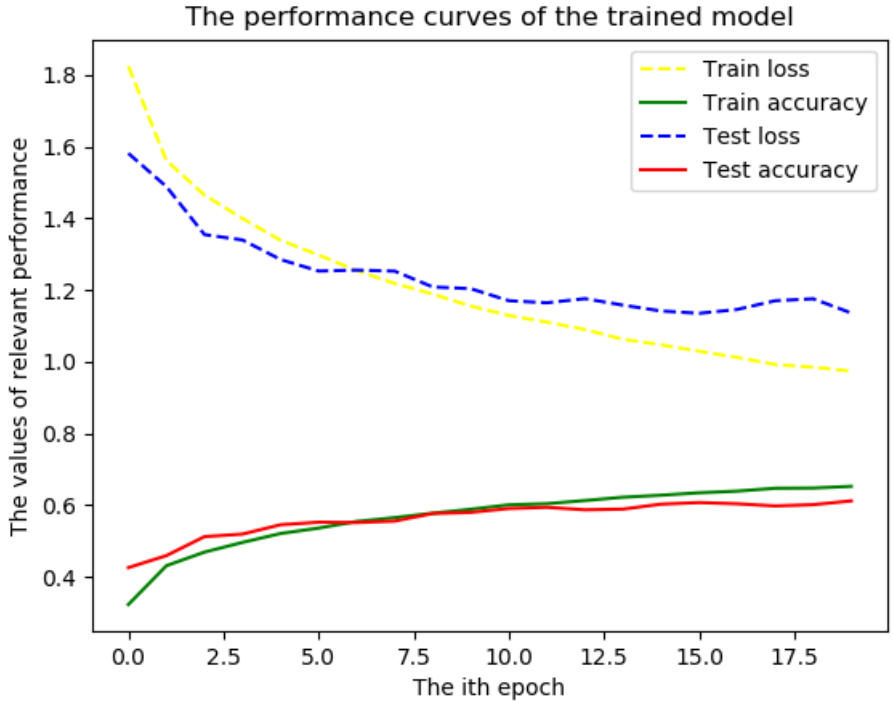


Fig. 1.8: The best performance curves among my experiments
when the initializer is "he_uniform", the learning rate is 0.02, the decay is 0.0001

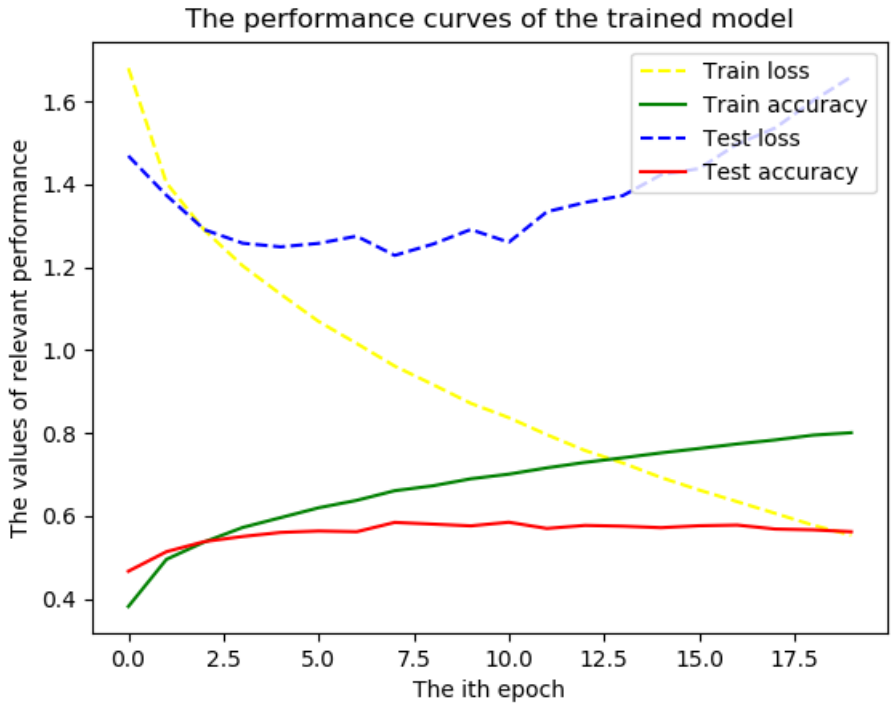


Fig. 1.9: The performance curves without “Dropout” parts
when the initializer is "he_uniform", the learning rate is 0.02, the decay is 0.0001

1.4 Discussion

My discussion of experiment results of Problem 1 is stated as below.

In part (a), I have read several papers from professor Kuo and some materials online, based on which I have written my understanding about CNN architecture for five questions. The details of my understanding can be accessible above.

In part(b), I have used Keras based on TensorFlow to build a simple convolutional neural network architecture derived from LeNet-5. When the initializer is “glorot_uniform”, the learning rate is 0.02, the decay is 0.0001, the accuracy performance curves’ plot is shown in Fig. 1.3. When the initializer is “glorot_uniform”, the learning rate is 0.01, the decay is 0.00001, the accuracy performance curves’ plot is shown in Fig. 1.4. When the initializer is “glorot_normal”, the learning rate is 0.03, the decay is 0.0002, the accuracy performance curves’ plot is shown in Fig. 1.5. When the initializer is “he_uniform”, the learning rate is 0.025, the decay is 0.0001, the accuracy performance curves’ plot is shown in Fig. 1.6. When the initializer is “glorot_uniform”, the learning rate is 0.04, the decay is 0.0003, the accuracy performance curves’ plot is shown in Fig. 1.7. The accuracy results ranging from the Fig. 1.3 to the Fig. 1.7 can be seen in the Table 1.

From the online source, I acquire the information that learning rate is a hyperparameter controlling how much we are adjusting the weights of the model with respect to the loss gradient while weight decay belongs to the weight update rule causing the weights to exponentially decay to zero. I believe the combination of those two parameters is more important than the initialization method so that I modified a bit to reach the accuracy of 0.6117 as shown in Fig. 1.8. One notable thing is that I have added the “dropout” layer, which refers to randomly ignoring

units (also known as neurons) during the training phase in order to prevent overfitting, between the fully connected layers from the Fig. 1.3 to the Fig. 1.8. As the Fig. 1.9 shows that if I delete the “dropout” layer, the train accuracy will be very high and the train loss will decrease without constraint, which is exactly the phenomenon of overfitting. There is no denying that my “best” performance mustn’t be the best one. The choice of optimizers and techniques of preprocessing the training dataset will also have a nonnegligible effect on the performance curves. I hope in problem2 I can realize higher accuracy.

In part(c), I will use the thinking from the paper[3] to make the VGG16, a very deep convolutional neural network famous for handling the ImageNet dataset, adapt to CIFAR-10 classification. The details of this idea’s implementation and the pros and cons of two approaches have already been discussed above.

References

[1] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324
[2] https://www.cs.toronto.edu/~kriz/cifar.html
[3] 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.
[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
[5] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.