Convolutional Neural Network for Handwritten Digit Recognition

Kuo, Shao-Heng, student number 201883361
Instructor: Professor Cha Eui Young
Department of Electricity and electronic Computer Engineering, Pusan National University

Abstract

Convolutional neural network (CNN) has been used widely and successfully in recent days. ILSVRC pushed CNN application ahead to object detection and pattern recognition field and leaded to many well-designed architectures within the past few years. In our work, we rebuilt several of the simple but conceptual CNN architectures which were proposed by Yann LeCun in 1989 [2]. The main goal of this paper is to compare the performance and learning speed of these architectures. We also preprocessed our input data set by operators Sobel, Prewitt, Laplacian and Gaussian Laplace, and then compared the performance of them. Due to the newly-made data set we used, we can also observe the performance of these architectures while dealing with data which is much more difficult to train compared to the original paper.

1. Introduction

Our research is relied on the paper which was published in 1989. Back to that era, when computer scientists were dealing with neural network problems, as a result of limited computational ability, most of them were trying to reduce the learning times. LeCun was one of those who emphasized the importance of generalization performance, and this concept frequently appears in his paper. Until these days, since CPU computational ability has progressed thousands times better than 1990s [1], and also the rising of GPUs, many complicated CNN architectures became feasible. The generalization performance also became a critical standard to evaluate the quality of a CNN architecture.

Convolutional neural network has several advantages compared to fully-connected neural network, for example, transformation invariance and weight sharing.

Transformation invariance refers to the unchanged output feature map due to the robustness to the slight input difference, for example, some little translations of an input image.

Weight sharing means we can extract a specific feature from input data by limited free parameters. The decrease in parameter number can save computational costs and prevent overfitting, which is often caused by too many free parameters.

Although CNN seems to be superior to fully connected neural network, we still need to realize that these advantages are based on our prior knowledge of the input data set. Take digit recognition for example, we know there might be some transformations while humans were writing them, so the shift invariance is reasonable for improving the performance. An example for explaining weight sharing's feasibility is the processing of finding a human being in photo images, by designing a set of weight sensitive to 'human' feature and using it over the whole image repeatedly.

As discussed above, in order to design a well-performance neural network with appropriate parameter number, we need to understand the property of our data set very well and take those properties into consideration.

2. Related Work

2.1. Handwritten Digit Data Set

Our data set consists of 480 digit samples, from 0 to 9, each of them has 48 samples. These samples are made evenly by four different people in 16 by 16 pixels. 320 of them are sorted into training data and the rest 160 samples are sorted into testing data randomly.

2.2. Methods Used in Networks

We used a modified hyperbolic tangent function as our activation function,

$$f(a) = A \tanh Sa$$

where $A = 1.7159$, which decides the amplitude of the hyperbolic tangent function. $S = \frac{2}{3}$, which refers to the slope at the origin. a is the output of units in our networks. The function is shown in Figure 1.

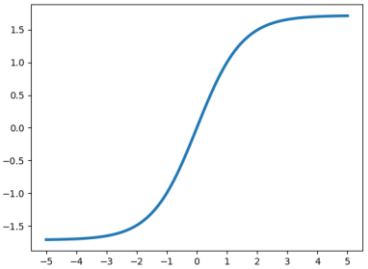

Figure 1. Modified hyperbolic tangent function

It can be observed that the value changing of the activation function will become almost zero when a is near absolute value of 5. This may sometimes lead to vanishing gradients issue during the process of back-propagation. This problem most happens in deeper neural networks, although our networks are at most two hidden layers (excluding input and output layers), we still have to avoid this problem. This can be done by choosing proper weights initial values,

$$minimum = \frac{-2.4}{F_i}$$

$$maximum = \frac{+2.4}{F_i}$$

the weights will be set as uniform distribution between *minimum* and *maximum*, where F_i is the fan-in of the unit, which is the input number of corresponding connections.

Uniform distribution between those values gives a safe initial weights values preventing from reaching the flat area of hyperbolic tangent function. If our value is too large after passing through activation function, the learning speed will be slow down severely.

For the loss function, we used Mean Squared error,

$$C = \frac{1}{P}\sum_{p}\sum_{o}\frac{1}{2}(D_{op} - X_{op})^2$$

where P is the number of training samples (we use P = batch size = 32 in our experiment), D_{op} is the ground-truth output and X_{op} is the predicted output by our networks.

3. Network Design

Our network was designed by the architectures shown in Figure 2.

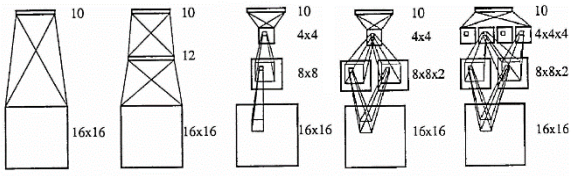

Figure 2. Five architectures [2] (net1 to net5 from left to right)

3.1. Net1-A Single Layer Network

This is a simple network with one input and one output layer, the output layer has 10 nodes and is fully connected with the input layer.

Number of free parameters: $16 * 16 * 10 + 10 = 2570$

3.2. Net2-A Fully Connected Network with One Hidden Layer

This network is built by adding one hidden layer with 12 units to net1. It is convinced that the additional layer will have the ability to extract more information from the input image.

Number of free parameters: $(16 * 16 * 12) + 12 + (12 * 10) + 10 = 3214$

3.3. Net3-A CNN without Weight Sharing

Net3 has two convolutional layers, both have depth value of one (It means there's only one channel in the feature map). The weights and biases of each output unit are different, so there's no weight sharing.

First hidden layer has filter kernel size 3 by 3, with stride 2. Second hidden layer has filter kernel size 5 by 5 and also has stride value 2. Since there's a downsampling process due to the existence of stride value not equal to one, the size of the network can be reduced, also some of the local information may disappear and result in transformation invariance.

Finally, the second hidden CNN layer is followed by a fully connected layer with 10 output units.

Number of free parameters: $(3 * 3 * 8 * 8) + (8 * 8) + (5 * 5 * 4 * 4) + (4 * 4) + (4 * 4 * 10) + 10 = 1226$

3.4. Net4-A CNN with Partial Weight Sharing

The most different part between net3 and net4 is that in the first hidden layer of net4, the CNN shares its weights by using same kernel to slide through the whole image. The other different part is that net4 has two channels in the first CNN layer. Noticed that there's no weight sharing in the second CNN layer.

But here, in contrast to modern CNN architectures which are mostly using tied biases, we used untied biased (so it shares same weight but has individual bias for each output). Untied biases is believed to perform better than tied bias when dealing with small size of training set [3].

Number of free parameters: $(3 * 3) * 2 + (8 * 8) * 2 + (5 * 5 * 4 * 4) * 2 + (4 * 4) + (4 * 4 * 10) + 10 = 1132$

3.5. Net5-A CNN with All Weight Sharing

Net5 is similar with net4, but there are four channels in the second CNN layers in net5. Both first and second CNN layers in net5 share weights and still use untied biases.

Number of free parameters: $(3 * 3) * 2 + (8 * 8) * 2 + (5 * 5 * 2) * 4 + (4 * 4) * 4 + (4 * 4 * 4 * 10) + 10 = 1060$

3.6. Training Process

We have tried both on-line update and batch update to train our data set. It came out that on-line update and batch update yielded similar accuracy. But thanks to the modern computational ability, batch update cost much fewer time to reach such accuracy. We used batch size 32 and recorded for 3000 epoches in every experiment.

With regard to the optimizer for updating weights and biases, we used Adam optimizer in every experiment, which indeed had a better performance compared to Gradient Descent optimizer.

4. Experiment Results

We built our network models by using tensorflow and thus some of our results will be presented by tensorboard.

4.1. Testing Results of Each Net

Table 1 shows the test accuracy on our testing data set after training for 3000 epoches.

Table 1: Testing accuracy

Network Architecture	Accuracy
Single Layer Network	57.5%
Fully Connected Network with One Hidden Layer	66.9%
CNN without Weight Sharing	67.5%
CNN with Partial Weight Sharing	73.8%
CNN with All Weight Sharing	75.6%

4.2. Results of CNN Architectures during Training

In Figure 3 and Figure 4, we recorded the accuracy and the loss for each CNN architecture by tensorboard, we can also observe the computational time of each model. The 'Step' in the horizontal axis means how many batches we have trained, as we use batch size 32, for 3000 epoches, there will be (320 / 32) * 3000 = 30k steps.

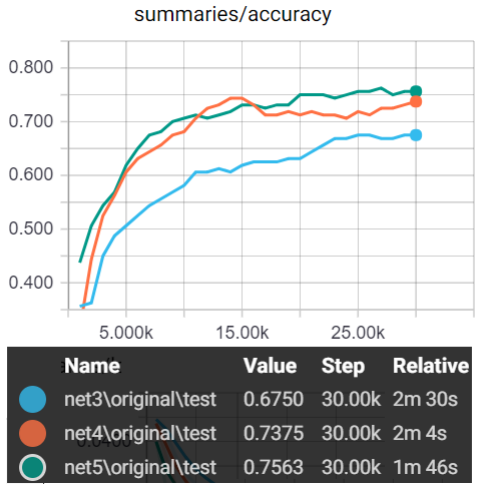

Figure 3. Test accuracy of net3, net4 and net5

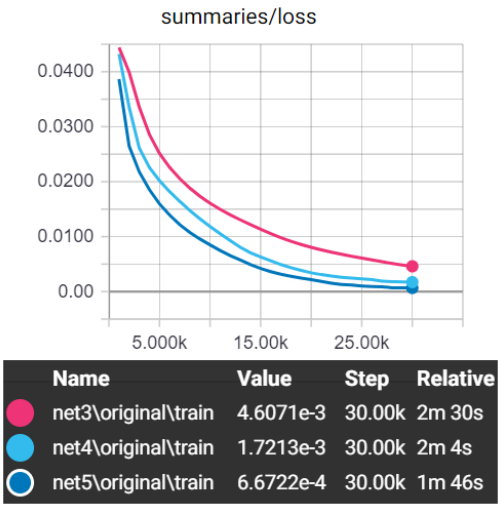

Figure 4. Training loss of net3, net4 and net5

4.3. Results of Net5 using Preprocessing Data

Figure 5 shows the accuracy of training data which is passed through several of different operators before entering network. Figure 6 shows their loss changing trend during the training process. We can find that after Sobel operating at only y-axis, it has a 4.3% increasing in accuracy compared to the original data.

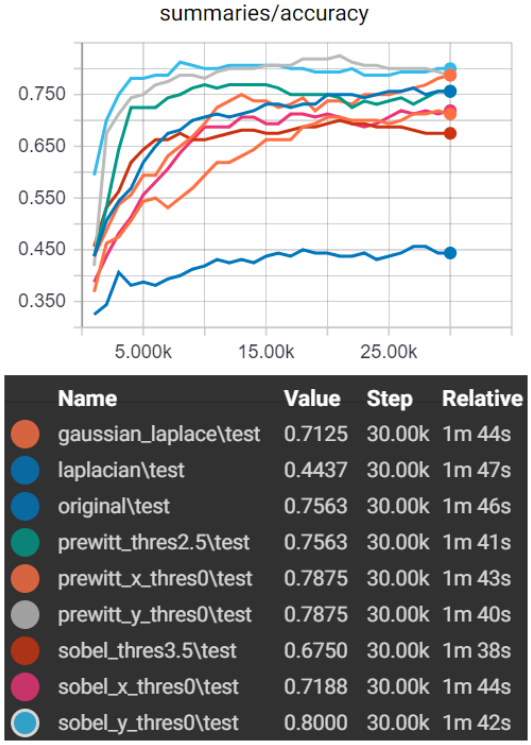

Figure 5. Test accuracy in net5 of each operator

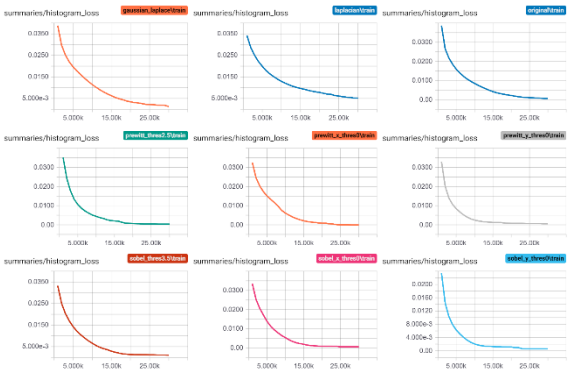

Figure 6. Loss trend in net5 of each operator

5. Discussion

5.1. Analysis of Learning Results

As shown in Table 1, we got 57.5% accuracy in single layer neural network, after adding one hidden layer with 12 nodes, the accuracy then achieved 66.9%, which is near ten percent of performance improvement. This shows that deeper networks may have better ability to extract more features of an input data.

We then focused on the comparison among net3, net4 and net5. As the training loss shown in Figure 4, three nets have decreasing loss during training process, this shows that there is no problem for them to learn the weights for fitting training data. Net5 got best accuracy performance 75.6% after 3000 epoches, which is 1.8% better than net4. But we can observe that the curve of net4 in Figure 3 reached a peak around 15k steps and then went down for about 10k steps, finally rose up to its final accuracy 73.8%. The peak has a value of 74.4%, which is better than the final result. This may be explained by overfitting issue since the training loss continued decreasing during the process.

With respect to the results in net5 while using different operators, we found that Sobel (80%) and Prewitt (82.5% at 20k steps) operating only on y-axis obtained significant better accuracy compared to the original data, but when using two-axis Sobel and Prewitt operator, there isn't any improvement or even worse performance. Gaussian Laplace got near four percent decreasing in performance while Laplacian operator seemed to lose the ability of fitting training data, as we can see the larger loss in Figure 6 compared to other operators.

5.2. About Batch Update

We know that if there is duplication in training data set, using batch update is not preferred due to the over-correction issue. While in our data set, which is made by four different people, the appearance of duplication might be less. This may explain why we obtained similar results with on-line update in our experiment by using batch update.

6. Conclusion

To design a proper network architecture, the prior knowledge about the data set stands an important role. In handwritten digit recognition problem, the characteristics of convolutional neural network, like transformation invariance and weight sharing nicely fit the purpose and lead to better performance compared to fully-connected neural networks.

Furthermore, the increased performance by using Sobel y-axis operator reflected the fact that horizontal edges (which can be detected by Sobel y-axis operator) might be a distinguishable feature in digit recognition. If we want to further improve the performance of digit recognition, this might be a feasible direction.

The number of free parameters of net3, net4 and net5 is 1226, 1132 and 1060 respectively. This shows that with fewer number of free parameters, we can still obtain better performance. Moreover, the computational times of each architecture shown in Figure 3 also correspond with the number of parameters, which shows that we got better performance and less computational time simultaneously by applying finer CNN architecture.

As described previously, our data set was made by four different people and thus it will be much more difficult to train. By using these CNN architectures, we got our best accuracy performance 82.5% (Net5 with Prewitt operator), which is not a satisfied result and much lower than nowadays standard. By observing the low-enough training loss, we concluded that our architecture has the ability to learn our training data completely but the testing accuracy didn't perform well. So this might be caused by the insufficient training data since our data set has much more diversity. In order to get better performance, amplifying our data set might be a probable way.

References

[1] D. A. Patterson, J. L. Hennessy, *Computer Organization and Design*, Elsevier, pp.42. 2012.

[2] Y. le Cun, Generalization and Network Design Strategies, June 1989.

[3] Harm de Vries, Tied Biases vs Untied Biases, March 2015.