ISSN:1226-7244 (Print)
ISSN:2288-243X (Online)
논문번호 18-01-03
14

j.inst.Korean.electr.electron.eng.Vol.22,No.1,14～20,March 2018
http://dx.doi.org/10.7471/ikeee.2018.22.1.14

Effects of Hyper-parameters and Dataset on CNN Training

Huu Nhan Nguyen*, Chanho Lee*★

Abstract

The purpose of training a convolutional neural network (CNN) is to obtain weight factors that give high classification accuracies. The initial values of hyper-parameters affect the training results, and it is important to train a CNN with a suitable hyper-parameter set of a learning rate, a batch size, the initialization of weight factors, and an optimizer. We investigate the effects of a single hyper-parameter while others are fixed in order to obtain a hyper-parameter set that gives higher classification accuracies and requires shorter training time using a proposed VGG-like CNN for training since the VGG is widely used. The CNN is trained for four datasets of CIFAR10, CIFAR100, GTSRB and DSDL-DB. The effects of the normalization and the data transformation for datasets are also investigated, and a training scheme using merged datasets is proposed.

Key words: Hyper-parameter, CNN, classification accuracy, weight factor, neural network training

---

* Dept. of Electronic Engineering, Soongsil University, Seoul, Korea.
★Corresponding author
E-mail: chlee@ssu.ac.kr, Tel:+ 82-2-820-0710,

※ Acknowledgment
This work was supported by the MOTIE (Ministry of Trade, Industry & Energy (10080568) and KSRC(Korea Semiconductor Research Consortium) support program for the development of the future semiconductor device. The EDA tools were supported by IDEC.
Manuscript received Feb. 20, 2018; revised Mar. 24, 2018; accepted Mar. 29, 2018

I. Introduction

Recently, convolutional neural network (CNN) is widely used in object recognition using deep learning due to its simple architecture and high classification accuracies [1]. The CNN training requires three essential inputs of a network architecture, a dataset and hyper-parameters such as a learning rate, a batch size, and initial weight factors. The hyper-parameters for the training process affect results of the training, that is, the weight factors. Unfortunately, since a set of parameters for optimization depends on the network model and the dataset, it is generally chosen arbitrarily or based on prior experiences and is adjusted by trial and error.

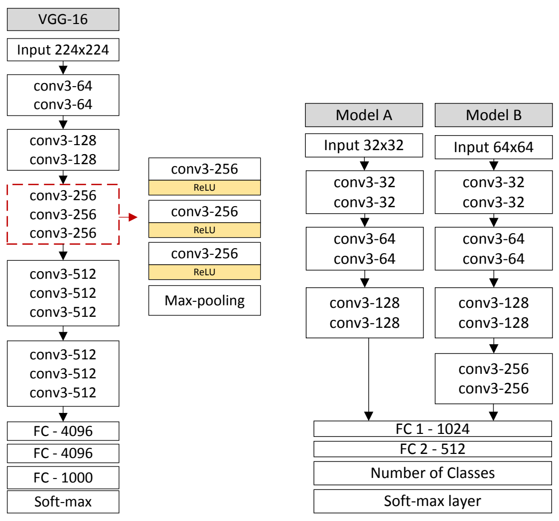
The learning rate (LR) initialization is the major problem in the gradient descent algorithm and its variants, which are the most popular in training CNNs, because it affects the convergence of CNNs [2, 3]. Training a deep neural network for a dataset with a large number of data often results in the 'out of memory' error. The batch gradient descent method that updates the weight factors after training the entire training dataset is expensive in both memory demand and computation [4]. The mini-batch gradient descent method performs better due to noises in the update direction allowing for jumping across the local minima [5]. However, the choice of good batch sizes depends closely on the network architecture and dataset, and is determined according to designer's experience. The initialization of weight factors affects significantly on training deep neural networks due to cumulative sum and big gradient magnitude at the final layer. After determining the hyper-parameters, the optimization algorithm affects the training process.

In this paper, we present the effects of hyper-parameters on training CNNs for various datasets to obtain optimal hyper-parameters and propose a training method to obtain a pre-trained weight factors for better training results. The effects of each hyper-parameter are analyzed in terms of the classification accuracy (CA) of the trained network and the training time for various datasets. The proposed method of obtaining pre-trained weight factors gives better training results by training a CNN with merged datasets first and then applying the weight factors to train the CNN for each dataset. It is useful when we have more than one dataset or other datasets with similar input dimensions are available.

II. Hyper-parameters for training CNN

A well-known scheme to initialize weight factors is loading the weight factors of a similar network trained with a complex dataset, and then re-training with the target dataset [6]. Inspired by the work, we merge two or more datasets including the datasets of interest to get a pre-trained set of weight factors.

The completeness of a dataset affects the training results of a CNN. Raw image samples that are included in the dataset may be diversified through lots of transformation algorithms such as scaling, cropping, horizontal or vertical flipping, and brightness or contract adjustment. A raw sample may be pre-processed according to a randomly selected transformation including no transformation when it is chosen for training. A



(a) VGG-16 (b) Proposed CNN

Fig. 1. Network architecture of VGG-16 and proposed CNN models.

dataset is augmented through the pre-processing which improves the training results of a CNN [7].

Raw image samples in a dataset include redundant information such as noises or outliers that deteriorate the training results [8]. The normalization reduces the fluctuation and weakens noises and outliers to increase the CA and to reduce the training time.

III. Experimental results

A. Experimental environments

The experiments were performed on a PC with 6GB GPU and Ubuntu 14.04 using Tensorflow library. In order to investigate the effects of hyper-parameters on training results, we design a CNN inspired by VGG-16. Fig. 1(a) shows the VGG-16 network of 5-convolutional layers for the input images with a resolution of 224x224. We modified the VGG-16 network for the input images of 32x32 and 64x64 resolutions as shown in Fig. 1(b). The model A and B have the same architecture except for the number of convolutional layers to process the images with different resolutions as shown in Fig. 1(b). The complexity of the proposed CNN is much simpler although it resembles the VGG-16.

We compared the proposed model with the VGG-16 in terms of the CA, the total number of weight factors and an inference time for 4 datasets shown in Table 1. The experimental results show that the proposed models are trained faster by 2-3 times with the smaller number of weight factors by 2.3 - 2.8 times than VGG-16 as shown in Table 2. The results have been scaled so that the proposed model has the value of 100%. The CAs are slightly smaller than those of VGG-16.

Table 1. Dataset summary

Dataset	No. of Training samples	No. of Test samples	No. of categories /image size	Objects
CIFAR10	50,000	10,000	10/32x32	Animals, transportations
CIFAR100	50,000	10,000	100/32x32	plants, household devices
GTSRB	39,209	12,630	43/64x64	German traffic signs
DSDL-DB	35,413	15,177	7/64x64	On-road objects.

B. Experimental results

Table 3 shows the effects of the initial LR on the CNN training. The first column represents the initial values of LR in range of [0.001, 0.1], the second and the third columns indicate the maximum CA and the number of epochs (NE), respectively, when the loss function becomes stable. The CA after training tends to increase as the initial LR is decreased from 0.1 to

Table 2. Comparison results of VGG-16 and proposed model

Dataset	Performance [%]	VGG-16	Proposed
CIFAR10 32x32	Accuracy	90.6	88.7
	Training time	220.0	100.0
	# of weight factors	243.4	100.0
CIFAR100 32x32	Accuracy	62.7	61.3
	Training time	219.9	100.0
	# of weight factors	243.3	100.0
GTSRB 64x64	Accuracy	97.7	97.2
	Training time	308.8	100.0
	# of weight factors	280.2	100.0
DSDL-DB 64x64	Accuracy	90.8	90.1
	Training time	303.7	100.0
	# of weight factors	238.0	100.0

0.01/0.005, and then they decrease slightly as the initial LR is decreased further. The initial LR values for the maximum CA depend on the characteristics of the datasets. The proposed network reaches the peak CA at LR=0.01 for CIFAR10 and GTSRB and at LR=0.005 for DSDL-DB and CIFAR100, respectively. When the initial LR is too large, the loss function tends to diverge because of the chain of big update, which blows up the total loss.

The batch size (BS) is the number of training samples for mini-batches as described in the previous chapter. The BS is chosen from 8 to 256 by doubling the previous number.

Table 3. Effects of learning rate in training

LR	CIFAR10		CIFAR100		GTSRB		DSDL-DB	
	CA[%]	NE	CA[%]	NE	CA[%]	NE	CA[%]	NE
0.1	54.6	160	16.8	120	34.7	30	63.0	155
0.05	79.2	280	43.3	140	72.5	45	83.0	190
0.01	88.4	215	56.4	130	96.0	40	88.4	180
0.005	87.4	290	58.2	185	95.5	45	89.6	195
0.001	85.3	410	56.2	300	93.5	50	86.0	205

Table 4 shows the effects of batch sizes in training. The values in the NE/TT columns are the number of epochs at which the CA becomes the maximum, and the training time (TT) in minutes corresponding to the NE. The CA increases fast as the BS is increased and becomes almost constant or slightly decreases as the BS is rise further. The batch size of 64 or 32 gives the best training results.

The loss functions for CIFAR10 and CIFAR100 with the BS of 8 started to diverge in the first epoch and the LR was decreased in the early stage. Practically, the networks for CIFAR10 and CIFAR100 with the BS of 8 were trained with very small LR and it took long time for the loss function to converge.

Table 4 also shows that it takes less NE for training as the BS decreases while the lengths of actual training time are not proportional to the NE. In SGD style training, the training time is dominantly consumed by data transfer between CPUs and GPUs for batch loading and returning gradient results and by computing time at GPUs while the time for loading raw data are necessary for the initial queue formation only. Besides, the number of weight factor updates with a small BS is larger than that with a large BS. Therefore, the data communication traffic for training networks with a smaller BS is heavier, and the training time per epoch increases as the BS is decreased. Therefore, the actual training time

Table 4. Effects of batch size in training

BS	CIFAR10		CIFAR100		GTSRB		DSDL-DB	
	CA [%]	NE/ TT[m]	CA [%]	NE/ TT[m]	CA [%]	NE/ TT[m]	CA [%]	NE/ TT[m]
8	73.5*	200/484	39.9*	170/178	81.2	4/8	84.7	74/19
16	83.5	43/52	43.7	128/73	89.1	10/11	86.5	94/15
32	88.4	132/45	56.4	140/43	96.0	20/19	88.4	110/13
64	89.5	145/35	58.4	205/41	96.1	25/22	87.9	120/15
128	87.6	175/37	56.9	215/44	96.1	40/35	87.2	150/17
256	86.9	200/40	55.1	255/51	96.1	45/39	86.9	19/23

decreases and then increases as the BS is increased in general. We find that the TT is the smallest for the highest CA except for GTSRB.

The weight factors are initialized to random numbers in normal distributions with various standard deviations (SD) in order to investigate the effects of initial weight factors. Fig. 2 shows the results when the SD is changed from 0.01 to 0.11. We failed in training the network when the SD was too small or too large because the weight factors seldom changed during the back-propagation. In the range of trainable SD, the CA tends to decrease or to stay at a similar value because small weight factors cause high gradients and have high chances to jump over many local minima by gradient updates.

Another way of initializing weight factors employs pre-trained ones. We merged DSDL-DB and GTSRB, and trained the CNN to obtain a pre-trained weight factors. The CNN was trained again for each dataset using the pre-trained weight factors. The CAs for DSDL-DB after training using the pre-trained

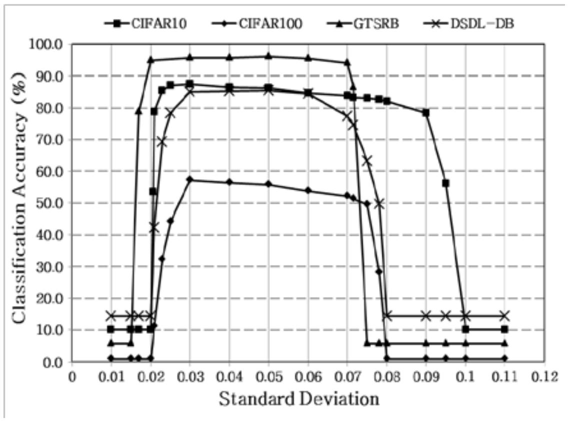


Fig. 2. Effects of standard deviation of normal distribution in weight factor initialization.

weight factors from the merged dataset are enhanced by 5% and 1.7% compared with the results using random numbers and GTSRB, respectively. On the other hand, the CA after training CNN for GTSRB using the pre-trained weight factors from DSDL-DB and the merged dataset is increased by 0.9% and 1.2%, respectively. DSDL-DB can be a superset of GTSRB as shown in Table 2 although the number of samples of traffic signs in GTSRB is larger than that in DSDL-DB. Therefore, the pre-trained weight factors from DSDL-DB can enhance the results of training for GTSRB while the opposite case does not work. However, the proposed approach does not work for CIFAR10 and CIFAR100. We presume the reason that the categories of the two datasets are unrelated, or the proposed network is not appropriate for CIFAR100 since the accuracies for CIFAR100 are much lower than those of the others.

The Fig. 3 shows the effects of data pre-processing of transformation and normalization in training the proposed CNN. The normalization removes the outliers, centralizing and scaling sample to clarify salient edges in signs. The training with the normalization improves the accuracies by 9.5 ~ 29.9% compared with the training using raw data. The training after the data transformation improves the accuracies by 13.9 ~ 31.4% compared with the training using raw data. The effect of the transformation is larger than the normalization except for the GTSRB. GTSRB already includes transformed data samples and the effect of transformation is weakened. If the normalization and the transformation are combined, the accuracies are increased by 17 ~ 40.1% as shown in Fig. 3. The results of training after the normalization and the transformation combined are better than the training with single processing for all datasets. The computation time for data pre-processing is negligible in training because the computation is made while queueing of data although the data pre-processing is very expensive computation.

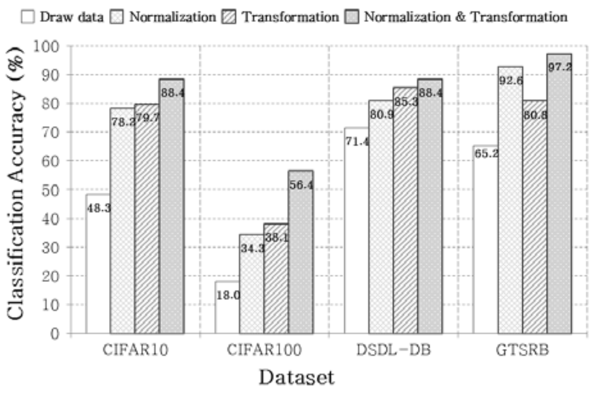


Fig. 3. The effects of data pre-processing of transformation and normalization examined in four cases.

Fig. 4 shows the loss functions according to training steps. The loss functions for CIFAR10 and CIFAR100 with the momentum converge much faster and the final values are smaller than those without the momentum. The CAs for CIFAR10 and CIFAR100 increase from 83.4% to 88.4% and from 48.1% to 56.4%, respectively. On the other hand, although the loss function of DSDL-DB with momentum decreases faster than that without momentum, the improvement is not as large as that of CIFAR10 or CIFAR100. Consequently, the increase of CA is not large (from 86.1% to 88.5%). The loss function of GTSRB without the momentum decreases fast enough and the loss function and the CA for GTSRB with the momentum are almost the same as those without the momentum. The additional training time per epoch due to the momentum is negligible (1.3-1.7s/epoch) compared with the total training time per epoch (10-50m/epoch). As a result, it is usually a good choice to train CNNs with the momentum.

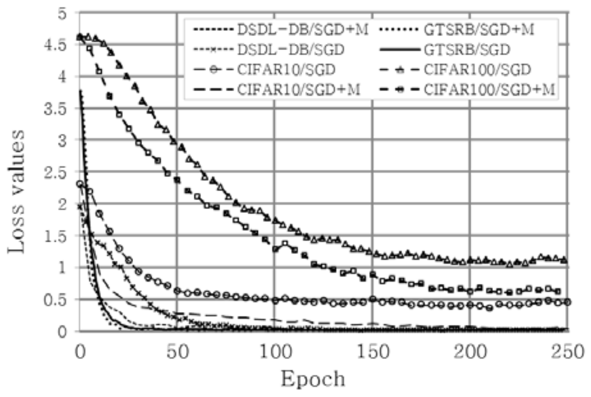


Fig. 4. Loss functions of training model by SGD and momentum algorithm.

Ⅳ. Conclusion

The effects of hyper-parameters on training the proposed CNN model are presented, and a training method to a training strategy for better training results is proposed. We obtain suitable initial hyper-parameters which are the initial learning rate in range of [0.01, 0.005], the batch size of 32 or 64, and the standard deviation of 0.05 with the momentum optimizer. The pre-trained weight factors obtained from a merged dataset are useful in many cases. The data pre-processing increases the classification accuracies greatly and need to be applied before training. The momentum optimizer accelerates training CNNs.

References

[1] K. Alex, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proc. of Neural Information Processing Systems*, pp. 1097-1105, 2012.

[2] Y. Bengio, “Practical recommendations for gradient-based training of deep architecture,” *Neural Networks: Tricks of the Trade*, Springer Berlin Heidelberg, pp. 437-478, 2012.

[3] T. M. Breuel, "The Effects of Hyperparameters on SGD Training of Neural Networks," https://arxiv.org/abs/1508.02788

[4] N. Ketkar, *Deep learning with Python*, Apress, 2017.

[5] M. Moller, “Supervised learning on large redundant training sets,” in *Proc. of Neural Networks for Signal Processing*, pp. 79-89, 1992.

[6] M.D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” in *Proc. of 13th European Conference on Computer Vision*, pp. 818-833, 2014.

[7] Y. Xu, R. Jia and L. Mou, G. Li, Y. Chen, Y. Lu, and Z. Jin, “Improved relation classification by deep recurrent neural networks with data augmentation,” in *Proc. of 26th International Conference on Computational Linguistics (COLING)*, pp. 1461-1470, 2016.

[8] A. Rusiecki, M. Kordos, T. Kaminski, and K. Gren, “Training Neural Networks on Noisy Data,” in *Proc. of 13th ICAISC*, pp. 131-142, 2014.

BIOGRAPHY

Huu Nhan Nguyen (Member)

2012: BS degree in Computer Engineering, University of Information Technology, Hochiminh, Vietnam.
2016: MS student in Dept. of Electronic Engineering, Soongsil University

Chanho Lee (Member)

1987: BS in Electronic Engineering, Seoul National University.
1989: MS in Electronic Engineering, Seoul National University.
1994: Ph.D in Electrical Engineering, UCLA
<Research Area> SoC on-chip-network, Memory controller, Image Pattern recognition