Utilizing of Casper Algorithm in CNN for Digit Recognition

Yujia Zhang

Research School of Computer Science, Australian National University

U6075459@anu.edu.au

Abstract. Convolutional neural network (CNN) has been successfully applied in digit recognition. Previous works on CNN usually use fully connected layers before the output layer, so the size of network should be provided prior to learning procedure, resulting in potentially oversized network. In this paper, Casper algorithm is applied in CNN proposing a constructive network architecture to add neurons incrementally and employing Progressive RPROP for compact network. Performance of the modified CNN is compared with the convolutional neural network of the same architecture with Lenet-5 on handwritten digits. Classification accuracy of the proposed architecture is almost identical to Lenet-5 architecture but in a gentler trend, while it reduces more than half of the hidden neurons and provides an adaptive solution. Further optimization and discussion are illustrated in final part.

Keywords: Convolutional neural network (CNN), Cascade, RPROP, classification

1 Introduction

Convolutional neural network has been shown to be powerful to process data saving in multiple arrays [1], applications include document reading and speech recognition. CNN takes advantage of convolution to guarantee sparse interactions, parameter sharing and equivariant representations, while utilizes pooling function to summarize statistic of the nearby outputs [2]. The architecture of CNN is sequence of stages, preserving the compositional hierarchical property of natural signals, including handwritten digits. Conventional CNN consists of multiple convolutional layers followed by a small number of fully-connected layers [3]. The fully connected layers dominate the majority of free parameters [4], requiring much memory access and thereby consuming power tremendously [5]. Moreover, the size of network should be provided prior to learning procedure [6], resulting in potentially oversized network.

A new network architecture as complementary for the fully-connected layers has been proposed, with narrow and deeply cascaded structure called Cascade-Correlation, also denoting as CasCor [7]. The topology begins with fully connected input and output units only, hidden units with maximum correlation value among the candidates are added one at a time. Implementing constructive algorithm, Cascade-Correlation ends with appropriate network complexity regarding to problem complexity [8]. Later, Casper algorithm is developed [9] as a variant of CasCor to improve generalization while reducing network complexity. It employs modified RPROP [10] rather than weight freezing to reduce the number of neurons. To guarantee the generalization, weight decay and loss decrease are taken into consideration [9] while adding new neurons.

The dataset used for evaluation is the MNIST database of handwritten digits [11]. The classification task takes grey images of handwritten digits as input with ten possible classes (0-9 respectively). Digit recognition is an essential part of document analysis, which has numeric applications in different industries and academia. It is widely used for evaluation among state of art algorithms and architectures, corresponding preprocessing and error rate are documented. So, a convincing performance comparison between the proposed architecture and conventional algorithms can be conducted by choosing MNIST database. Moreover, the essential preprocessing has been achieved, the work in this paper can be focused on optimizing architecture to reduce computational complexity while maintaining optimal accuracy.

This report begins with description of data set and implementation of the classical CNN architecture, Lenet-5. And the architecture optimization is also based on Lenet-5, substituting the fully connected layers with cascaded structure. The report also presents the performance comparison between the modified network and the original Lenet-5 and other state of art architectures. Reasons for comparison results and further discussions are also provided.

2 Method

2.1 MNIST Dataset for Training and Testing

The MNIST data set [11] is a classic dataset for learning techniques and pattern recognition methods on real-world data. It is separated into four files, two of them are images and labels for training set, which contains 60,000 instances. And the other two files are for testing set of 10,000 examples. One of the motivations for utilizing MNIST in the proposed CNN

architecture is that the images are preprocessed and formatted, so the performance would only be influenced by the architecture modification, rather than the uneven distribution between training set and testing set.

Pytorch DataLoader has encapsulated regular MNIST dataset. The images in the dataset are of size 28 * 28, and the central 20 * 20 area contains the shrunk binary image of handwritten digit. The folder named "data" is specified as the root directory of dataset, in which "raw" directory contains the four files mentioned before and "processed" directory contains training.pt and test.pt. Training set and testing set could be loaded by specifying "train" parameter to "true" and "false" respectively. And in both of them, the plt images are converted to tensor after which normalization is applied, in that way, the background pixels (white) are of value -0.1 and foreground pixels (black) are of value 1.175. So, the distribution of input is roughly with mean of 0 and variance of 1, accelerating the training procedure. Batch sizes for training on classical CNN, training on CNN with cascade architecture and testing are specified in command line arguments: "--pretrain-batch-size", "--finetune-batch-size" and "--test-batch-size" accordingly. If there is available GPU, CUDA pin memory would load copy of the tensor and data is loaded in parallel.

Different classifiers have been applied on MNIST, with test error rate documented. Lenet-5 is considered to be one of the classical CNN architecture, and its test error rate on MNIST is 0.95%. In the rest of the essay, a CNN of the same architecture with Lenet-5 is implemented as baseline. And the performance of newly proposed architecture is compared with these state of art algorithms.

2.2 Implementation of Lenet-5 for Benchmark and Encoder

Build a convolutional neural network of the same architecture with Lenet-5 using a customized Module subclass. There are six layers in total between the input image and the output which specifies the digit class the input belongs to. Lower layer is responsible for some elementary features and the subsequent layers are used to extract higher-order features. The first layer is a convolutional layer, which is composed of 6 feature maps. Each 5 * 5 neighborhood area in input, denoting as respective field, corresponds to the same 25 trainable coefficients and 25 trainable bias in one feature map. Utilizing identical weight vector, which is called kernel in convolutional network, in one feature map for different parts of image is based on the principle of sharing weights. Then every unit in the feature map is the convolution result of a respective field in the previous layer and the corresponding kernel. Different kernels are used in different feature maps in order to detect different local features. In Lecun's implementation of Lenet-5 [12], the input images are of size 32 * 32, so every feature map is of size 28 * 28 after the convolution with 5 * 5 kernel. But the regular MNIST dataset built in pytorch is composed of 28 * 28 input images. To guarantee the same architecture and consequent results as in Lenet-5, paddings of size 2 are implemented during convolution calculation, so the 6 feature maps in the first convolutional layer are also of size 28 * 28.

Subsampling is implemented on feature maps in the first layer, the results compose the second subsampling layer. For each feature map in first layer, every non-overlapping 2 * 2 neighborhood area is averaged, multiplied by a trainable coefficient and added by a trainable bias. The results are activated by sigmoid function and documented in the second layer, which is a subsampling layer and is of size 14 * 14. And for digit recognition, once a local feature has been detected, its precise location is irrelevant for recognition, because the feature would present in different positions for different instances. Sharing weight eliminates the influence of the different positions in various instances and sub-sampling reduces spatial resolution. So, the successive layers of convolutions and sub-sampling reduce resolution but result in redundant representation.

Third and fourth hidden layer in Lenet-5 are convolutional layer and subsampling layer, with 16 feature maps. Like the first two layers, 5 * 5 kernels are used for convolution, so the resulting feature maps in third layer are of size 10 * 10, and non-overlapping 2 * 2 areas are squeezed, so feature maps in the fourth subsampling layer are of size 5 * 5. However, the implementation in this essay is simplified from the classic Lenet-5. In Lenet-5, most of units in the third layer only connect to three of the feature maps in previous layer to control the number of connections and to break symmetry. In the implementation of this essay, every feature map in third layer is connected to every map in the second layer, which partly accounts for the performance decrease.

The following two layers work as fully connected layers in this scenario. The fifth layer implements convolution with 5 * 5 kernel to form 120 feature maps. Because the feature maps in fourth layer are of size 5 * 5, the fifth layer is simplified as typical fully connected layer with 120 neurons in our implementation. And there is another hidden layer with 84 neurons before output.

Testing the trained convolutional neural network using test data, print accuracy as benchmark result. Plot loss curve to describe learning procedure for evaluation part. And the first four layers serve as encoder for visual features in the proposed architecture below.

2.3 Construct Convolutional Neural Network Employing Casper Algorithm

2.3.1 Architecture of the Network

The structure optimization from classical Lenet-5 is to replace the fully connected network before output with cascade network. During training on Lenet-5, determine parameters in the first four layers. And tuning is implemented on the cascade network with the parameters in the previous layer frozen. In this way, the successive convolutional layers and

subsampling layers worked as encoder to describe visual features of each input, and the cascade network is the classifier to output the digit class. The overall architecture is illustrated in Fig.1

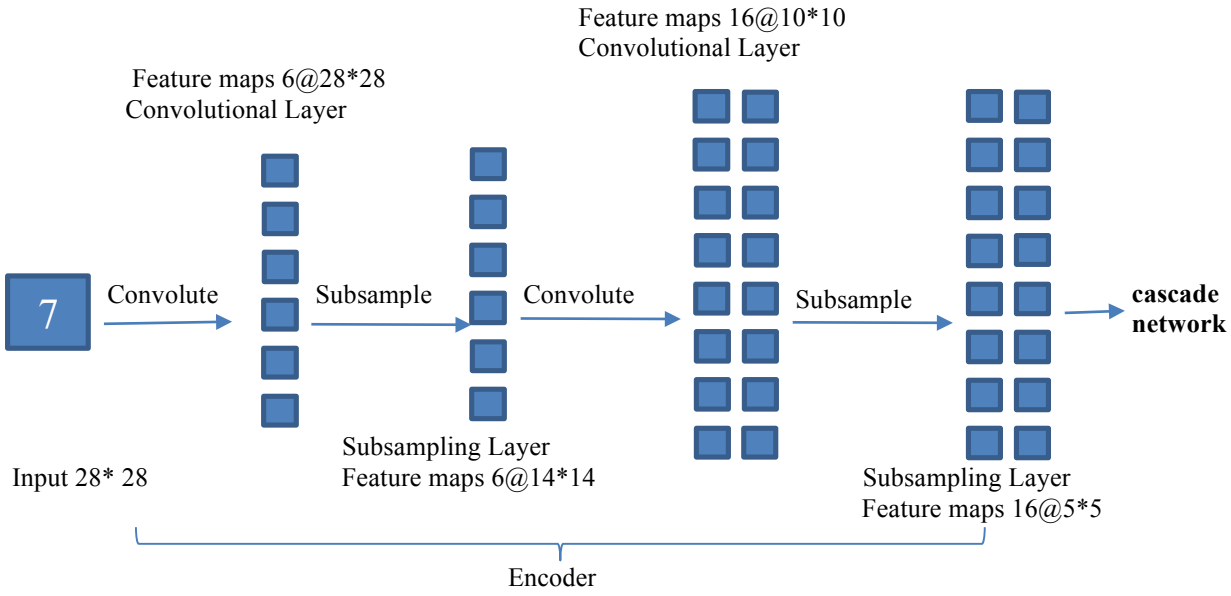

Fig. 1. Topology of the proposed architecture, each plane represents a feature map

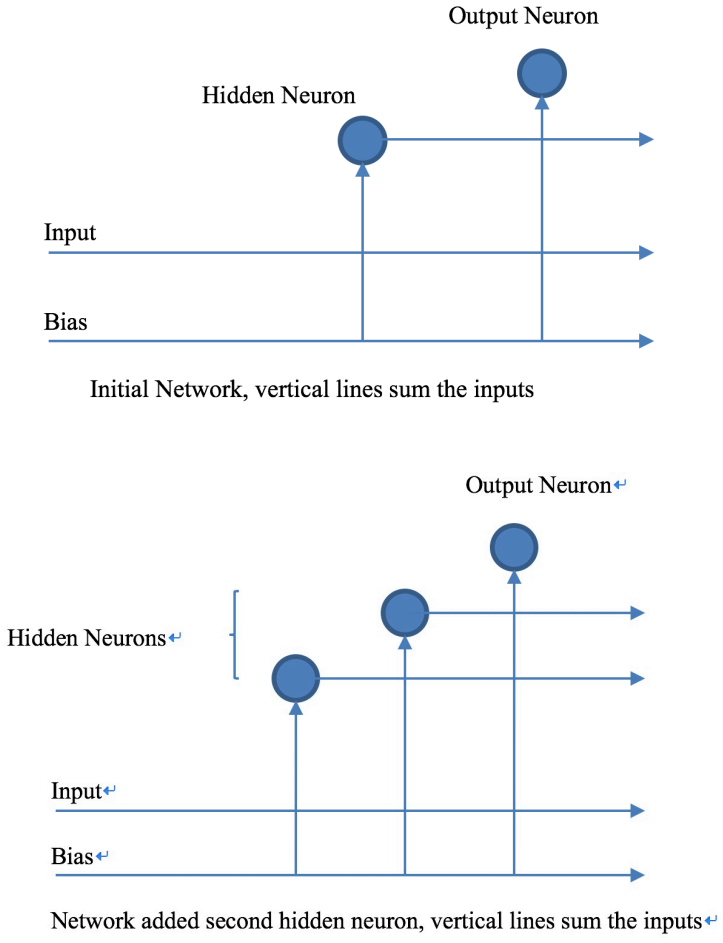

Fig. 2. Topology of the cascade network part

The cascade neural network is described as follows. It is feedforward and obtains its topology and weights during the learning process [13]. It starts with a single hidden neuron, and adds successive hidden neurons one at a time [9], as in

Fig.2. The architecture can be seen as network with hidden layers, each of which only contains one neuron. For each hidden layer (actually a hidden neuron), inputs are encoding features from previous layers together with outputs from hidden neurons added before. So, the number of inputs for hidden layers increases at one layer by layer. The number of output for hidden layers remains one, because there is only one output neuron. Compared with fully connected architecture, in which the number of neurons in output layer is the same with the number of classes, complexity for connection is decreased dramatically in cascade topology. An upper bound number of hidden neurons is 300, and there are more than 600 (400 + 120 + 84 + 10) hidden neurons in Lenet-5, so in that way the decrease of computation complexity could be proved clearly.

2.3.2 Training Methodology

The training customs the forward function to define a forward pass based on the architecture in constructor. The first four layers are trained in the same way as in Lenet-5, and a cascade network is applied after that. The training of Lenet-5 is pre-train process, determining the parameters for the first four convolutional layers and subsampling layers. Once the training of cascade network has begun, the parameters in the encoder part are frozen. The inputs passed to every neuron in the cascade network are the input from encoder and intermediate results of the previous hidden neurons. Use sigmoid function as activation function and adjust it into [-0.5, 0.5], then pass the result to the proper output connection. Sum the outputs of each hidden neuron as the classification result.

A progressive RPROP is employed as optimizer with standard parameter value [10] to train the cascade network. Different learning rates are assigned to weights based on the position in the cascade network as in Fig.3. The L1 Weights contains weights connecting new neuron with input from encoder or intermediate results of the previous hidden neurons. L2 Weight refers to output of new neuron and other weights are of L3 learning rate. L1, L2 and L3 are 0.2, 0.005 and 0.001 respectively. In implementation, parameters are saved in a list, so when a new neuron is added, the list add new parameter in while popping the old ones out to update the relative position in the network. L1>>L2>L3 makes it possible for new neuron to learn the remaining error and reduce the overall error like using correlation value, and for old neurons to change weights that are suboptimal due to greedy policy but in small learning rate. Instead of weight freezing in CasCor, the modification eliminates oversized network without too much inference from other neurons.

New neuron is added when the train loss decrease is not significant, that is, compared to the maximum loss in last three epochs, the train loss is not decreased by 0.99 by default (and this threshold for loss drop can be modified in command line argument '--loss-drop-ratio-thres'). There are necessary but insufficient conditions for adding neuron, one of them is that the added neuron should make train loss drop at least 1% (and this could also be specified by '--loss-drop-ratio-thres' argument) [10]. Another one is if the same architecture has been maintained for more than 80 epochs (this could also be modified in command line argument '--wait-max-steps-add-neuron'). But this necessary condition will not be true unless the architecture has not been modified for 20 epochs (this could also be modified in command line argument '--wait-min-steps-add-neuron'). Instead of using correlation value to determine adding neuron, this network can be generalized without saturation problem. Weight decay is performed by 0.9 every step to guarantee generalization.

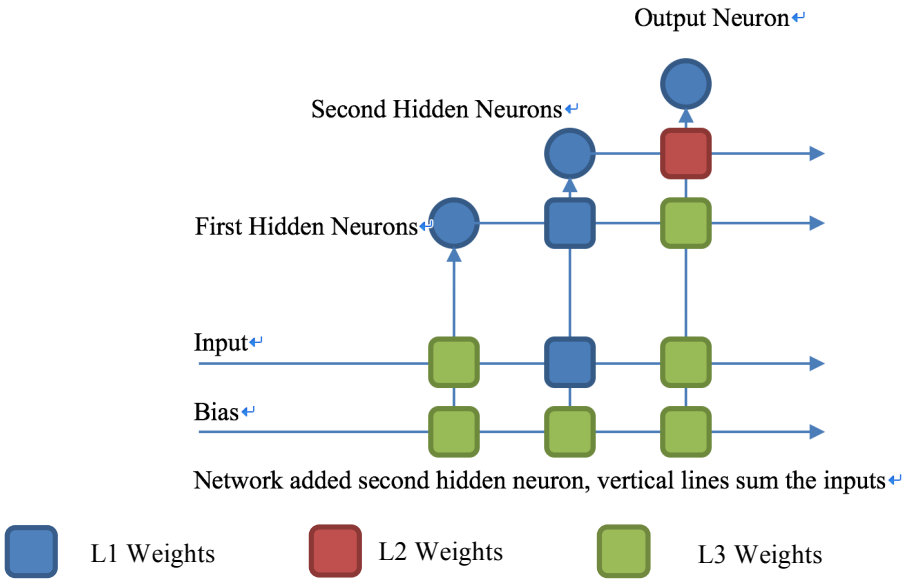

Fig. 3. Different learning rates for weights based on the position in the network

The parameters in training procedure could be specified to overwrite the default values while running the minst.py script. '--pretrain-batch-size', '--finetune-batch-size' and '--test-batch-size' specify the batch sizes used for training in Lenet-5, which is actually used as pre-training in the proposed architecture, the batch size to train the cascade network and batch size for testing. '--pretrain-epochs' and '-- finetune-epochs' specify how many epochs are iterated when training the Lenet-5 for encoder and training the cascade network. '--lr' specifies learning rate for stochastic gradient descent optimizer for Lenet-5 training. '--loss-drop-ratio-thres', '--wait-max-steps-add-neuron' and '--wait-min-steps-add-neuron' are listed before as the hyper parameters while adding new neurons to the network. '--use-pretrain' is set false by default, the pre-trained model files may not be loaded successfully when the script is running for the first time because of different environment settings. But if the script is excuted for more than one time, this setting could be set to true to save time for pre-training, and only training for cascade network would be implemented. To save the pre-trained model and the final model after tuning by cascade network, '--model-path' and '--fine-tune-model-path' could be modified as directory to save the model files, and they are located in "models" folder by default.

The log during training has been documented to plot curves when plot-loss.py runs. "finetune_log_300" is the original log file while pre-training and tuning, and the other two log files are created by the script. Folder named "data" as described before with raw data and preprocessed data, and folder named "models" saves model files for pre-training and training result with cascade network.

3 Results and Discussion

Evaluation is performed in two ways: accuracy to evaluate the overall performance, poisson negative log likelihood loss as loss function and its plot figures during training and testing. Accuracy is chosen for comparison with the other sate of art algorithms, because it is referred in the MNIST document as the performance description. Poisson negative log likelihood loss is used, because the output activated by softmax is a vector with possibility for each class and the measurement describes the probabilistic confidence.

3.1 Comparison between Lenet-5 and the CNN with Casper algorithm

The benchmark network is of the classical architecture of Lenet-5. But its performance is inferior to Lecun's implementation [12]. There are two possible reasons for the performance decrease. One is that the inputs for Lecun's experiment are images of size 32 * 32 but the built-in MNIST dataset for pytorch is composed of 28 * 28 images. The increase of total size of the image while the digit part remains the same makes it possible for potential distinctive features to locate at the center for the highest-level feature detector, thereby increasing accuracy in Lecun's experiment. Another possibility is that there is symmetry break in Lecun's implementation via partial connection between two layers, but in the implementation of this essay, the connection is from every respective fields in the previous layer to every unit in the next layer.

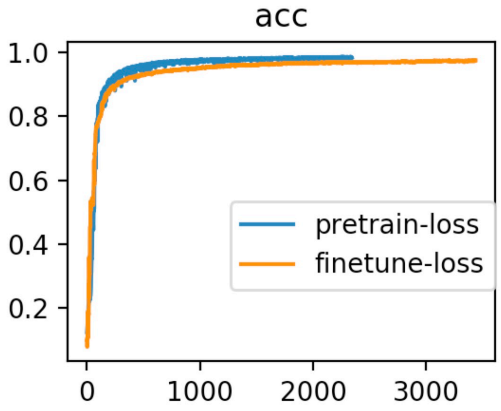

Fig. 4. Accuracy and for classical Lenet-5 and CNN with Casper algorithm

Accuracy curves for classical Lenet-5 (pretrain-loss curve) and CNN with Casper algorithm (finetune-loss curve) are presented in Fig.4. The CNN with Casper algorithm achieves accuracy in a gentler way, while there are intensive fluctuations for Lenet-5. And the proposed architecture decreases the number of neurons by more than half during learning procedure, because the 300 neurons are added incrementally. The number of connections is reduced even dramatically, because in fully connected network, there are 10 output neurons and every hidden neuron before output layer should connect to each of them, but in cascade network, there is only 1 output neuron. The performance of CNN with Casper is achieved by weakening back propagation of errors, since in cascade network, learning occurs almost in one layer by using

different learning weight while tuning [10]. Given that most of free parameters are contributed by the fully connected layers [4], the cascade topology reduces computation complexity greatly while maintaining the performance. But classical Lenet-5 achieves slightly higher testing accuracy than the proposed architecture in the final stage, 98.3% and 97.5% respectively. The reduction of neurons weakens the representation ability for the classifier after the encoder, accounting for the performance reduction in the final stage. And the different trends for accuracy increase are mainly observed at the later epochs during training, which indicates only adjusting parameters for one layer in cascade topology may be harmful for further optimization because the suboptimal learning results in previous epochs should be correlated by neurons added later. Besides, the batch size for training Lenet-5 is 64 while that of the proposed architecture is 1024, which partly results in the accuracy decline. The curve for tuning the cascade network is longer than that of the Lenet-5, because the terminal condition is determined by the convergence of the two algorithms, which indicates that it may take longer for cascade network to learn.

Poisson negative log likelihood loss plot for CNN with Casper algorithm and classical architecture of Lenet-5 are illustrated in Fig.5. The proposed algorithm decreases loss in a gentler way, and intensive fluctuations happen in Lenet-5. The training losses for both of the network are almost identical, but Lenet-5 achieves slightly lower testing loss in the end. Apart from the greatly decrease in the number of neurons, the greedy policy to decrease loss during training also accounts for the potential overfitting. There are less free parameters in the proposed topology, yet more hyper parameters are proposed, including the threshold of loss drop determining when to add new neurons, maximum and minimum steps after which new neuron is added. These hyper parameters could influence the performance because they could determine the severity of adding neurons, thereby influencing the overall complexity. There should be a tradeoff between computation complexity and performance optimization. In further implementation, if performance upgrades are more desirable, the conditions for adding neurons could be relaxed to achieve more complex network with stronger representation ability.

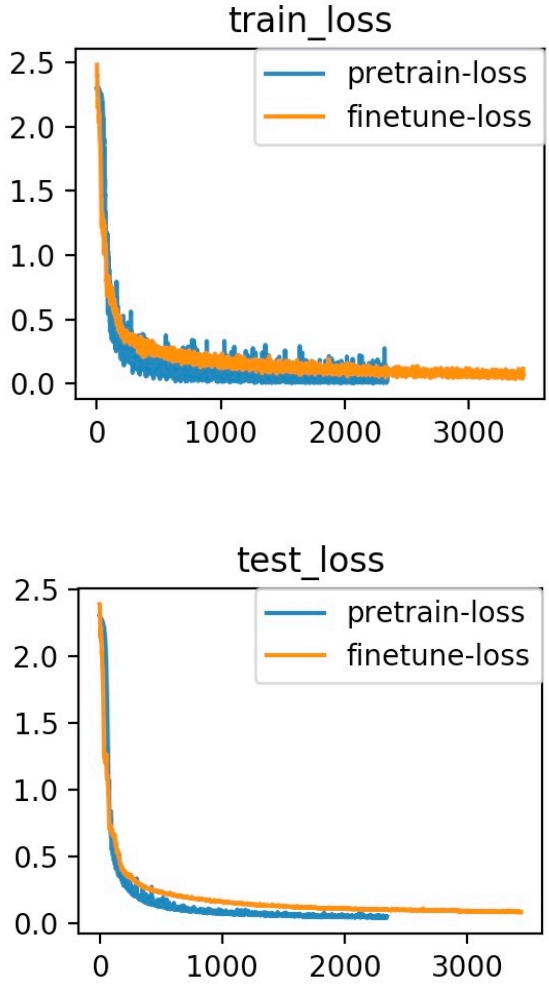

Fig. 5. Plot of loss decrease during training(above) and testing (below)

3.2 Comparison between CNN Utilizing Casper and Other Methods

Test accuracy is the used to describe performance between Casper and other methods, because it is the measurement documented in [11]. The proposed topology achieves 2.5% error rate with the default settings of the hyper parameters. It outperforms the linear classifiers and K-Nearest Neighbors without pre-processing, because the deep neural network is capable of representing non-linear visual features [14]. The performance is better than classic neural network without too many hidden layers or hidden neurons, because the weight sharing and sub-sampling eliminate the computation complexity and chance of over-fitting. And it also outperforms some non-linear classifiers like PCA or RBF, because in these algorithms, there should be many pre-defined features and rules, which reduce the generalization in the big dataset.

It outperforms Bayes [15] and decision tree algorithm [16], because it avoids the downsides of these algorithms. Bayes classifier makes strong assumption of conditional independence [17], but when the assumption is violated, the accuracy cannot improve continuously with the growth of training instances. Overlapping in decision tree results in oversized decision surface, and the performance is highly dependent on the initial design [18]. There is no prior assumption or initial topology required in Casper algorithm, eliminating early asymptote as in Bayes or instability in decision tree.

4 Conclusion and Future Work

A new architecture of CNN with constructive cascade network is implemented. The convolutional layers and sub-sampling layers are employed as encoder, and the cascade topology is classifier to get the final output class based on the feature encoded by previous layers. The pre-train procedure is identical to the classical Lenet-5, but the parameters in previous layers are frozen during the learning procedure of the cascade network. Hidden neurons are added incrementally that each neuron acts as a layer, there is no need to define the network topology prior to learning procedure. Optimizations based on CasCor are proposed: modified RPROP rather than weight freezing to achieve compact network, weight decay and loss decrease are taken into consideration while adding new neurons to guarantee the generalization.

Performance comparison is conducted between Lenet-5 and the proposed architecture. The classification accuracy is almost identical and the proposed network achieves that in a gentler way, however, only half neurons are needed in new architecture and number of connections is reduced even dramatically, indicating decrease of training cost and topology complexity. The result proves that, constructive algorithm like Casper contributes to compact topology whose size is appropriate for the complexity of the problem [7].

However, the optimization may have downsides. The neurons are added one by one during learning procedure, so it may take longer to learn. This can be proved by the length of curve for the two algorithms until convergence. There are more hyper parameters in the architecture and the performance could be vulnerable to the setting of these parameters.

There is also work for further optimization. The acasper algorithm [19] selects model automatically while constructing and regularizing network, providing better generalization. More dataset should be tested on the proposed architecture to see if it is expandable to other computer vision tasks. And to determine whether the number of hyper parameters could be cut down, further work should find the internal connections between topology complexity and performance optimization.

References

1. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436.
2. Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press.
3. Y. LeCun et al., "Backpropagation applied to handwritten zip code recognition", *Neural Comput.*, vol. 1, no. 4, pp. 541-551, 1989.
4. Z. Yang et al., "Deep fried convnets", *ICCV*, pp. 1476-1483, 2015, [online] Available: http://arxiv.org/abs/1412.7149.
5. S. Han et al., "EIE: Efficient inference engine on compressed deep neural network", *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, pp. 243-254, Jun. 2016.
6. Dhole, V. S., & Patil, N. N. A Review of Cascade Correlation Neural Network for Software Cost Estimation.
7. S.E. Fahlman, C. Lebiere, The Cascade Correlation Learning Architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, 1990.
8. Köppen, M., Kasabov, N., Coghill, G., & Springer-Verlag. (2009). Advances in neuro-information processing: 15th international conference, ICONIP 2008, auckland, new zealand, november 25-28, 2008, revised selected papers, part I. New York: Springer.
9. Treadgold, N. K., & Gedeon, T. D. (1997, June). A cascade network algorithm employing progressive RPROP. In *International Work-Conference on Artificial Neural Networks*(pp. 733-742). Springer, Berlin, Heidelberg.
10. Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Neural Networks, 1993., IEEE International Conference on*(pp. 586-591). IEEE.
11. Yann.lecun.com. (2018). MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [online] Available at: http://yann.lecun.com/exdb/mnist/ [Accessed 25 May 2018].
12. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, november 1998.
13. Kwok, T. Y., & Yeung, D. Y. (1997). Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE transactions on neural networks*, *8*(3), 630-645.
14. Bishop, C. :Neural networks for pattern recognition. Oxford University Press, Oxford (1995)
15. Meng, H., Appiah, K., Hunter, A., & Dickinson, P. (2011, June). Fpga implementation of naive bayes classifier for visual object recognition. In Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on (pp. *123-128). IEEE.*
16. *Marée, R., Geurts, P., Piater, J., Wehenkel, L., Hong, K. S., & Zhang, Z. (2004). A generic approach for image classification based on decision tree ensembles and local sub-windows. In Proceedings of the 6th Asian Conference on Computer Vision(Vol. 2, pp. 860-865). Asian Federation of Computer Vision Societies (AFCV).*
17. Russell, S. J., & Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.
18. Safavian, S. R., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, *21*(3), 660-674.
19. Klutchkov, K. (2013). Neural Networks. [online] Available at: http://courses.cecs.anu.edu.au/courses/CSPROJECTS/13S2/Reports/Kroum_Klutchkov_Report.pdf [Accessed 29 Apr. 2018].