Handwritten digits classification using an auto-encoder and network pruning technique

FanYU[1]
E-mail: u5872700@anu.edu.au

[1] FanYU, Research School of Computer Science, Australian National University,
108 North Road Acton ACT 2601 Canberra, Australia

Abstract. This paper builds a convolutional neural network (CNN) classifier that classify handwritten digits. The original images are compressed using a three-layer auto-encoder which is evaluated by the trained CNN classifier. The auto-encoder and the fully-connected layer of CNN are simplified using network pruning technology and its effectiveness is evaluated. The result shows a good accuracy of the CNN with 98% correct prediction on the input images. The result also shows that compression and pruning techniques can reduce the network and the image size as well as the complexity of the network without losing accuracy.

Keywords: Convolutional neural network (CNN), Handwritten digits classification, Auto-encoder, Network pruning

1 Introduction

Handwritten digit recognition is an important problem in optical character recognition [1]. For many years, it has been used as an important application for pattern recognition and machine learning algorithms [1]. Standard databases have been built to set a standard for techniques comparison. MNIST database of handwritten digits is a free standard database which contains 60000 training images and 10000 test images [1]. MNIST is a relatively simple database for researchers to try new techniques.

Neural networks have been used to perform classification and regression tasks and are suitable for handwritten digits recognition. However, a traditional feed-forward fully-connected neural network trained using back-propagation method can be slow to train [2]. On the other hand, features (pixels) from original data set may be similar or identical to each other, then the image compression technology can be used to compress the images in order to reduce the complexity of classification neural network and the space to store the dataset. The size of the hidden layer of compression network (an auto-encoder) could be used to determine the degree of compression [2]. For initial value, a rather large number could be used to build the hidden layer, then any redundant units should be removed. This technique is called network pruning [2]. Many properties of neurons such as distinctiveness, relevance, contribution, sensitivity and badness could be used to identify the redundant units [2].

In this paper, we will build a convolutional neural network (CNN) to learn features from MNIST handwritten digits images then try to classify them based on the features learnt. A three-layer feed-forward auto-encoder with 400 initial hidden units is built for the purpose of compression. We will use the preprocessed MNIST training dataset to train the CNN classifier and the auto-encoder and use the preprocessed MNIST testing dataset to test the effectiveness of them. After that, we will prune the auto-encoder and the fully-connected layer of CNN based on distinctiveness property of hidden neurons and test their effectiveness against original classifier and compressor.

2 Background

2.1 The MNIST dataset

The MNIST handwritten digits dataset comes in two parts. The first part is the training set which contains 60000 handwritten digits images [6]. The second part is the testing set which contains 10000 handwritten digits images [6]. Each image in MNIST dataset is 28 by 28 pixel in size and has a label that indicate its correct classification (which digit it actually is) [6]. The images are greyscale (only one channel) and are the scanned handwriting samples taken from a set of 250 people [6]. This dataset is included in many deep learning frameworks such as pytorch and tensorflow.

2.2 Images compression and network pruning

The compression algorithm is done by an auto-encoder which is also known as auto associative neural network [4]. An auto-encoder is a feed forward network with the target that the output of the neural network will be identical to input neural network [4]. In the middle of the network is the hidden layer that do a reduction of the dimension of the input features (pixels), this is where the images are compressed. The weight from inputs layer to hidden layer performs as the compression function and the weight from hidden layer to output layer performs as the decompression function [5]. A typical auto-encoder architecture shown in Fig 1 [5].
The hidden layer of the compression auto-encoder may contain redundant or less important hidden neurons which can be further pruned from the network. This process can improve the generalization performance and can reduce the dimension of compressed images. Many attributes can be used to detect redundant neurons, such as relevance, contribution, sensitivity, badness and distinctiveness [2]. The distinctiveness of hidden neurons is measured by the output vectors for each hidden neuron. Output vectors are formed by activation outputs of the hidden neurons with all the input patterns in training set. The pair-wise angles of output vectors then can be calculated and any two neurons with angles ranging from 0-15° are considered similar and with angles ranging from 165° to 180° are considered complementary [2]. Both kinds of neurons are the target for pruning.

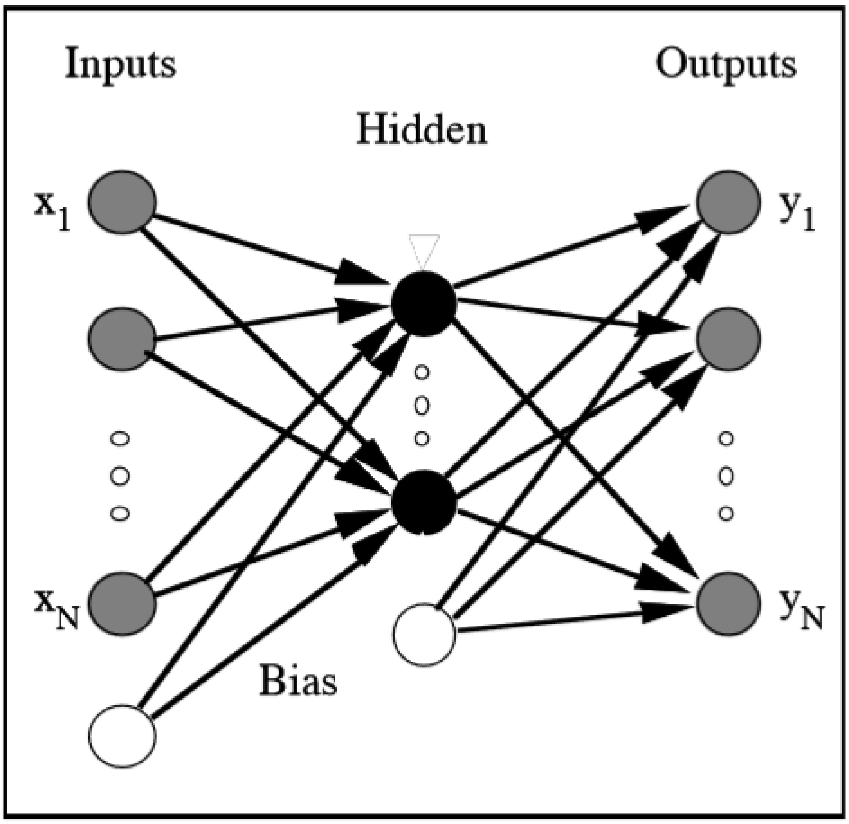

Fig. 1. A normal feed forward auto-encoder [5]

3 Method

3.1 Data set preprocessing

The MNIST dataset is included by our deep learning framework so we can directly load it. We create three different subsets of the dataset. The first subset containing 60000 data items (images) with the batch size of 128 and is used for training both the auto-encoder and the CNN. The second subset containing 10000 data items (images) with batch size of 1000 and is used for testing the effectiveness of the auto-encoder and CNN. The third subset contains the 10000 data items from the testing set but it has the batch size of 10000 and is used to get the activation output vector so we can prune the auto-encoder and the CNN. For each subset, the data items are shuffled so we can train and test our networks in a random way.

3.2 Classification

A CNN is constructed to be the classifier. It contains two convolution plus pooling layers to extract features from the input 28 by 28 image. Then a three-layer fully-connected feed-forward neural network is built to transform the feature maps produced by the convolution and pooling layers to 10 outputs which represent the log probability that the input was one of the digits 0 to 9. The first convolution layer uses 1 input channel and 10 output channels with the kernel size of 5. The first pooling layer uses the max pooling method with stride 2. The second convolution layer has 10 input channels and 20 output channels with the kernel size of 5. The second pooling layer is the same with the first one. Both convolution plus pooling layers use ReLU as their activation function. The fully-connected neural network has 320 input neurons, 50 hidden neurons and 10 output neurons. It also uses the ReLU activation function. The final output vector of the network is modified by the Log SoftMax function so the maximum output' index represents the predicted digit. The loss between the output vector and the target is produced by Negative Log Likelihood loss function to update the whole network with an SGD optimizer. The network is trained by the training set with batch size of 128 for 10 epochs and then tested by testing set to see the effectiveness of classification. Both the loss calculated by the loss function and the accuracy (number of data items correctly predicted / total data items) will be used to evaluate the classifier. The structure of the CNN is shown in Fig 2.

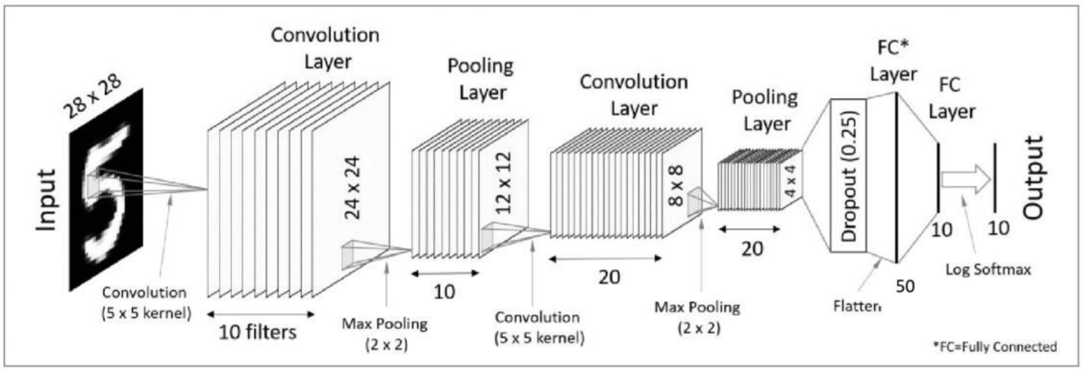

Fig. 2. The basic structure of the CNN

3.3 Compression

A three-layer feed-forward auto-encoder is constructed to be the compressor. All connections are from units in one level to next level. As discussed in section 2.2, the auto-encoder has equal number of input neurons and output neurons of 768, corresponding to 768 pixels (28 by 28) of each data item. The hidden layer represents the compressed version of images and initially has 400 hidden neurons, they use sigmoid function as their activation function, so the activation outputs are constrained between 0 to 1. The input data of images is flattened to be compatible with the auto-encoder and the decompressed output of the auto-encoder is refactored to the original image format.
The auto-encoder uses MSE between the decompressed images (outputs) and original images (inputs) as loss function with the target that the outputs are identical to the inputs (MSE equals 0). The auto-encoder uses Adam as its optimizer. The network is trained by training set with batch size of 128 for 10 epochs and then tested by testing set to see how well it compresses and decompresses data. The MSE loss is used to evaluate the effectiveness of compression. The comparison between decompressed images and the original images will also be used to evaluate the effectiveness of compression.

3.4 Network pruning

The trained auto-encoder discussed in section 3.3 is pruned using the algorithm proposed in section 2.2. The pruning dataset is fed to the trained auto-encoder and the output activation vector is calculated and normalized to 0.5 for each hidden neuron (the output vectors have values between 0 and 1 as discussed in section 3.3). The angles between each vector are calculated and any two neurons with angle smaller than $30°$ will be considered too similar so that one of them is removed and its weight vector is added to the remaining one (In our implementation, the neuron with smaller index is removed). Similarly, any two neurons with angle larger than $150°$ will be considered complementary and both of them will be removed. It is worth noticing that we are not using $15°$ and $165°$ as threshold because no neurons can be pruned when under such threshold. In our implementation, the removed neurons have their weight vectors and bias set to 0. All the output neurons also set the weight connected to the removed neurons to 0. In this way, the removed neurons have no effect to the whole auto-encoder, thus, they can be considered removed.

The fully-connected network in the CNN is also pruned using the same technique. The activation vector of the hidden layer of the network is used to calculate the angles and identify the similar or complementary neurons. The same methods are used to remove the neurons.

3.5 Evaluation

The experiments done in this paper are discussed as follow.
An auto-encoder is firstly trained by training data set and then tested by testing data set. During the training process, we output the second batch of decompressed images for every epoch to see how the auto-encoder performs during the training.
The CNN is then trained by training data set and tested by testing data set. The average loss and the accuracy of the CNN when classifying the testing data set will be used as baseline to compare with result using auto-encoder and pruning technique.
Then we compress and decompress the testing data set using the trained auto-encoder. The decompressed images are fed to the trained CNN to be classified and the results will be compared with the baseline to evaluate the performance of the auto-encoder.
The trained auto-encoder then goes through pruning process and redundant hidden neurons are removed as discussed in section 3.4. Then the simplified auto-encoder is used again to compress and decompress the testing data set. The simplified decompressed testing data set is fed to the trained CNN. The loss and accuracy will be used to evaluate how well is the compression and decompression when multiple similar and complementary neurons are removed from the auto-encoder.
The trained CNN then goes through pruning process and redundant hidden neurons in fully-connected layer are removed. Then the pruned CNN is used to classify the decompressed images produced by the pruned auto-encoder. The loss and accuracy will be used to evaluate how well the CNN does the classification when multiple similar and complementary neurons are removed from the fully-connected layer.

4 Results and discussion

As discussed in section 3.5, 5 experiments have been performed.
Experiment #1 trains the auto-encoder for 10 epochs, for each epoch, the decompressed image of second batch is compared with original image to evaluate the effect of compression and decompression.
Experiment #2 trains the CNN using training data set and test the network with testing data set.
Experiment #3 compresses and decompresses the testing data set using the trained auto-encoder. The decompressed version of testing data set is fed to the CNN to do the classification.
Experiment #4 prunes the auto-encoder and then compresses and decompresses the testing data set. This new version of testing data set goes through the trained CNN to do the classification.
Experiment #5 prunes the CNN and then classifies the decompressed version of testing data set from pruned auto-encoder.
The result of experiment 1 is shown in Fig 3.

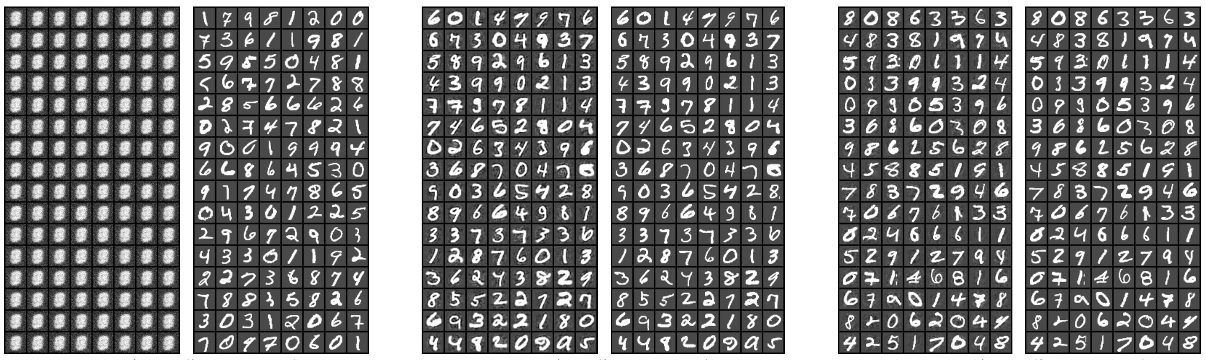
Epoch = 1(loss=1.045) Epoch=2(loss=0.062) Epoch=10(loss=0.012)

Fig. 3. Original images(right) versus decompressed images(left) from auto-encoder

At the beginning of training, the digits can't be restored and barely the shape of digits has been learnt by the auto-encoder. At the second epoch, the decompressed images are a little bit blurry but very similar to the original digits. At the final epoch, the images can be compressed and decompressed very well by the auto-encoder.
The result of experiment 2-5 is shown in Table 1.

Table 1. Overall results for experiments 2-5 done in this paper

Experiment number	Average Loss	Accuracy
2	0.0665	9795/10000(98%)
3	0.0673	9793/10000(98%)
4	0.0681	9796/10000(98%)
5	0.0697	9782/10000(98%)

The result from the experiment 2 shows that with 10 epochs training, the CNN can classify the digits very well, only 2% of the images are mispredicted. In experiment 3, as the decompressed images from the auto-encoder are almost identical to the original ones, the accuracy for classification only drops about 2 out of 10000 when using the decompressed version of testing set produced by the auto-encoder. This further indicates that the auto-encoder can compress and restore the images with very little loss.
The results from experiment 4 shows that even if we remove redundant or similar hidden neurons in the hidden layer of the auto-encoder, the quality of compression and the decompression remains the same as the loss and accuracy are the same level with the results produced in experiment 3. The same conclusion can be drawn for the pruning process for the CNN. As shown in results for experiment 5, after we have pruned the fully-connected layer of the CNN, the loss and the accuracy still maintain the same level.
We compare the classification result with other methods using the same data set as shown in Table 2 [3].

Table 2. Result for other approaches to do the handwritten digits classification using MNIST [3]

Method	Classification Error rate
Linear Classifier	8.4%
Nearest Neighbor Classifier (K=3)	2.4%
Pairwise Linear Classifier	7.6%
Large Fully Connected Multi-Layer Neural Network (5)	1.6%
LeNet1	1.7%
LeNet4	1.1%

The two linear classifiers achieve about 91-92.4% of accuracy, which is significantly lower than our CNN approach. The Nearest Neighbor Classifier achieves about 97.6%, which is close to our implementation. The Large fully-connected MLP and the first CNN approach LeNet1 are roughly at the same accuracy as our CNN approach. The LeNet 4, however, achieves much higher accuracy because it is much more complex with 260000 connections and 17000 free parameters [3].
Overall, the effectiveness of our CNN and auto-encoder is quite good.

5 Conclusion and future work

Initial experiments show that a CNN with two convolution plus pooling layers can do a good job in handwritten digits classification, providing a new choice for automated handwritten digits classification.
While a complex network may take long time to train, an auto-encoder can perform well in compressing images and reducing network complexity while still maintain the representativeness of the images.
For auto-encoders, the compression rate may be hard to decide. Then the network pruning technique can detect the redundant neurons in autoencoders, and we can remove these neurons to make images even more compressed.
The same idea holds for the CNN too. The fully-connected layer of the CNN may also contain redundant neurons. The network pruning technique can also apply to it to reduce the complexity of the CNN.

Other researches have been performed on auto-encoder. For a normal feed-forward auto-encoder, the first layer of weights may implement a non-invertible compression function. This problem can be addressed by shared weight topology where the weight between input and hidden layers are identical to weight between hidden and output neurons [5]. This is an improvement to our implementation and will be investigated in the future.
Besides the normal fully-connected auto-encoder, the CNN itself can become an auto-encoder because convolution and pooling can reduce the size of the image and extract features from it. Using a CNN instead of a fully-connected network as an auto-encoder is another interesting topic to study in the future.

References

[1] D. Li, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]", IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, 2012.

[2] G. TD and H. D, "Progressive image compression", *Neural Networks, 1992. IJCNN., International Joint Conference on*, vol. 4, pp. 403-407, 1992. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/227311/

[3] Y. LeCun, L. D Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U.A. Muller, E. Sackinger, P. Simard and V. Vapnik, "Comparison of learning algorithms for handwritten digit recognition", International Conference on Artificial Neural Networks, pp. 53-60, 1995. [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/lecun-95b.pdf

[4] M. Scholz, "NLPCA - nonlinear PCA - auto-associative neural networks - autoencoder bottleneck neural networks - Matthias Scholz", *Nlpca.org*, 2018. [Online]. Available: http://nlpca.org/

[5] G. T.D., "Stochastic bidirectional training.", *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on.*, vol. 2, pp. 1968-1971, 1998. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/728185/

[6] M. Nielsen, "Neural Networks and Deep Learning", Neuralnetworksanddeeplearning.com, 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap1.html