Feature Compression Technique for Neural Networks in Letter Recognition Tasks

Yinshuo Bai[1]

Research School of Computer Science, Australian National University
u5968211@anu.edu.au

Abstract. In this paper, we focus on two types of English letter recognition tasks: spoken English letter recognition and handwritten letter recognition. We employ fully connected neural network approaches to resolve the spoken English letter recognition task on a real word datasets, and we use convolution neural network to resolve the handwritten letter recognition task on EMNIST dataset. For both tasks we apply the same feature compression technique to reduce the size of neural networks. We conduct extensive experiments to examine the effectiveness of our compression technique. We observe a significant improvement on training time and the size of neural network while keep the performance at same level. For spoken English letter recognition task, we report a test set accuracy of 94.23% for our neural network performance and the compression technique reduces the training time by 20.95%; for handwritten letter recognition, we report a test set accuracy of 92.34% and the training time reduced by 30.40%.

Keywords: Letter recognition · Convolution neural network · Network reduction.

1 Introduction

Letter recognition is a fundamental task in machine area. Techniques of recognizing letters from speech signals or images can be furthered used in recognition tasks for extracting words or sentences from speech or image data. Automatic speech recognition is one of the challenge tasks in the field of machine learning. It is the foundation of many intelligent systems, such as voice search, speech-to-text applications and automated simultaneous interpretation. In this work, we focus on the task of automatic recognition of spoken English letters. The spoken letter recognition task is non-trivial and even challenging due to the fact that the phonetic similarity among some English letters, for example, 'B' vs. 'P' vs. 'D'. In this task, we focus on the automatic spoken English letter recognition task. Handwritten letter recognition is an essential technique which provides machine the ability to retrieve text information from images. For both letter recognition tasks, the recognition algorithms should be resistant to the possible transformation or variance in real world data caused by the personal differences, and therefore can be quite challenging for traditional algorithms.

A number of works employ neural networks to address the character recognition problems [2, 7, 17, 14, 15]. While the use of deep neural networks, such as LeNet-5 [10], a type of Convolutional Neural Networks (CNN), greatly improves the accuracy of the task, the computational performance, for example the training time, becomes another emerging issue for training a neural network, especially for the deep neural networks. Some techniques have been proposed to reduce the training time of a convolution neural network. For example, pooling, weight sharing and feature selection [18, 16]. In this work, we employ an input compression technique to reduce the size of neural networks and further reduce the training time of networks. We apply our input compression technique to the *spoken* and *handwritten* letter recognition tasks, and we also conduce extensive experiments to examine the effectiveness of our compression techniques.

For spoken English letter recognition task, first we employ a fully connected feed-forward neural network with two hidden layers to resolve the recognition task and examine its performance on the dataset we choose. Then, we also employ a neural network compression encoding technique to reduce the size of neural network while keep the recognition performance at a similar level. For handwritten letter recognition task, we employ a convolution neural network with two convolution layers and two fully connected layers to resolve the recognition problem, and test the model on EMNIST [4] dataset. Similar to the spoken letter recognition task, we apply the same input compression technique to reduce the size of network and the training time.

We implement our neural network and encoder with PyTorch and perform extensive experiments to examine the performance of our networks. For spoken letter recognition task, we report that our fully connected feed-forward neural network achieves an average accuracy of 93.59% on the UCI's ISOLET spoken letter recognition dataset [5], which is slight lower then the accuracy reported on the same dataset with similar approaches by [6]. We also report that our compression encoder reduces the size of our recognition network by 84.64% (from 729 units to 112 units)

and reduces the training time by 20.95% while keep the recognition performance at a similar level. For handwritten letter recognition task, we report our test accuracy of 92.64% on EMNIST letter dataset. The result outperforms the result of 85.15% by using a fully connected neural network, which is reported by [4]. We also report that by applying the input compression, the training time of network reduces by 30.40% (from 92.03 seconds to 64.05 seconds) while the test set accuracy is kept at a similar level.

2 Methods

The letter recognition task can be modeled as a multi-class classification task. For spoken letter recognition task, the predictor receives a feature vector (of length 617, in the ISOLET dataset we use) as an input and output a prediction vector of length 26 with each dimension of the output vector represents an English letter. For handwritten letter recognition, the input feature is a gray scaled image ($28 * 28$ in EMNIST dataset) and the output is the same to spoken letter recognition task.

We now introduce the structure of our neural network for the spoken and handwritten letter recognition tasks and describe the approaches we employed to reduce the size of neural network to improve the computational performance.

2.1 Feed-forward Neural Network for Spoken Letter Recognition

We construct a simple fully connected feed-forward neural network for the spoken letter recognition task. Our network contains two hidden layers with 64 and 32 hidden units respectively. The input layer contains 617 input units and the output layer has 26 units. We use Rectified Linear Units (ReLu) [13] as activation functions in both hidden layers and the input layer. The loss function we choose is cross entropy error function. The choice of cross entropy error function is based on the classification nature of our recognition task. We perform a mini-batch [9] training to train our network to introduce noisiness considering the non-convex property of our task but still keep acceptable computational performance. The mini-batch size is set to 20. We train our neural network on the ISOLET dataset for 50 epochs with a learning rate of 0.01. We note that although the neural network we construct for spoken letter recognition task is straightforward comparing to modern neural network structures which have achieve good results on chronological data, for example RNN [12], the network achieves an acceptable accuracy of 93.59%.

2.2 Convolution Neural Network for Handwritten Letter Recognition

We use a convolution neural network (CNN) for the handwritten letter recognition task. The network contains two convolution layers followed by two fully connected layers. The fist convolution layer uses 16 $7 * 7$ kernels and the second convolution layer uses 32 $7 * 7$ kernels. After each convolution layer, we use a average pooling layer to sub-sample the output data. The following two fully connected layers receives the outputs of the second convolution layer and map the output to 4096 and 26 dimension vectors respectively. The output layer contains 26 units (1 for each letter). Similar to the spoken letter recognition, we use ReLu as activation functions for convolution layers and cross entropy as error functions. However, for the fully connected layers we use sigmoid activation functions. The reason is that though ReLu can avoid gradient vanishing problem, it still introduces other issues to training. For example, the dead neurons [11] and bias shift [3] problems. For the fully connected layers near the output, sigmoid activation functions can avoid the shortcomings of ReLu and will not cause too much gradient vanishing. The mini-batch size is set to 100 because EMNIST can provide sufficient data now. We train our neural network for 30 epochs with a learning rate of 0.01. This networks finally achieves an acceptable test set accuracy of 92.34%.

2.3 Compression Encoder for Reducing Feature Dimension

We now introduce our compression encoder for reducing the dimension of input feature vectors and the size of neural network for the recognition task. We notice during the experiments with our recognition network that our original structural design of the neural network used for spoken letter recognition contains more then 600 units on at the input layer while the first hidden layer contains only 64 hidden units and the other hidden layer contains only 32 hidden units. Another observation comes from the manual inspection on EMNIST dataset. We notice that the while each image ($28 * 28$) is gray-scaled and contains a letter at its center, it still contains a lot of unnecessary information except for those we are concerned (the letter itself in our task). For example, we can resize the image to $20 * 20$ and still recognize the letter shown by the image. These observations imply that variables in feature vectors are highly correlated and contain a large amount of redundant information. Though our neural network solution are

resistant to this redundant, which means this will not introduce a accuracy performance issue, a high dimensional input vector will increase the size and complexity of the neural network and may cause a significant computational issue.

With the intuition that the dimension of input feature vectors can be significantly reduced, we design and implement a encode to compress the input vectors and use a simplified neural network with the encoded feature input vectors to do the spoken letter recognition task. We first introduce the encoder we use to compress the feature vectors.

Compression Encoder The idea of compressing the input feature vectors to reduce the complexity of neural networks comes from the image compression work in [8]. In [8], an image of 64×64 is divided evenly in to 16 non-overlapping patches with the same size and each image patch is represented by a vector of length x. The original image then can be represented by a vector (or matrix) with $16x$ elements. The key idea of compression in [8] is to find a low dimensional representation for each image patch, i.e. to reduce x.

The compression is accomplished by a neural network with 256 (16×16) input, a hidden layer with x hidden units and an output layer with 256 units. The compressor (the neural network) is trained with each image patch as a single input and at the same time, as the output. When the training has finished, we can easily compress an image patch by use the image as input to the network and the value of hidden layer (size of x) is the compressed representation of the input image patch. This compression technique helps reduce the complexity of the image in [8] and we use the same technique in our letter recognition tasks to reduce the size of input feature vector and the complexity of our recognition neural network.

Compression Encoder in Spoken Letter Recognition Task We now introduce the structure of our encoder used in spoken recognition task to reduce the dimension of input feature vectors. For implement convenience, we construct a neural network with two parts: the encoder and the decoder. The encoder corresponds to the input layer and the hidden layer of the compressor in [8] while the decoder corresponds to the hidden layer and the output layer of the compressor.

The encoder constitutes of an input layer of size 671, the original size of feature vectors of ISOLET dataset, a hidden layer of length 200 and then an output layer of length 64. The decoder uses a reversed structure of encoder: input layer of size 64, a hidden layer of size 200 and the output layer of size 671. The compressor can be considered as a network connecting the encoder part and the decoder part: it uses the output of encoder network as the input of decoder network. To train the compressor, we split the training dataset and uses 20% of training data to train the compressor. This part of data then will not be used in the afterwards training and testing of the recognition neural network.

Compression Encoder in Handwritten Letter Recognition Task The encoder we use in handwritten recognition task is similar to the one we introduced previously. We use two parameters: *input_size* and *target_size* to control the size of an encoder. The *input_size* indicates the width of image we want to compress while the *target_size* is the width of image after compression. We note that since we are processing images in this task, the input to encoder will be first flattened to a vector before being fed to the encoder and the output of encoder will be reshaped before again used by a CNN. For this task, we randomly pick 5% of instances from training dataset to train the encoder, and as we did in spoken letter recognition task, the instances used to train the encoder will not be used in neither training or testing of CNN.

2.4 Neural Network with Reduced Features for Recognition Tasks

We now introduce our approaches to apply the compression method to reduce the dimension of the input feature vectors and the modification we make to the original neural network used in letter recognition tasks and describe the whole training and testing procedure with this compression extension.

Feature Reduction for Spoken Letter Recognition

- **Compressor training**
  The first step is to train the compressor. We randomly pick 20% instances from out training dataset and remove them from the training set. We use these data instances as the input and output of our compression encoding neural network (the compressor). We then train our compressor by mini-batch method with the same mini-batch

size of 20 for 150 epochs. After the compressor is trained, we run separately the encoder part of the compressor on the remaining training data as well as the test set. As we discussed in previous sections, the encoder part of a compressor can be used as a standalone neural network which takes the original representation of a feature vector as input and the output of the encoder part gives the compressed low dimensional representation of the original feature vector.

– **Simplified Neural Network for Recognition Task**
Since the input feature vectors are compressed into a low dimensional representation, we now can reduce the number of units in input layer and simplify the structure of the original neural network. Our simplified neural network contains only one hidden layer with 32 hidden units, which is just the second hidden layer in our original network. The simplified network has 64 inputs, which is the dimension of our compressed representation for features, and the output of the simplified neural network remains to be 26 because we have 26 classes for our classification task.

Feature Reduction for Handwritten Letter Recognition

– **Compressor training**
The training of encoder is similar to the process described in Section 2.4. We first randomly pick 5% instances from out training dataset and remove them from the training set. We use these data instances as the input and output of our compression encoding neural network (the compressor). We then train our compressor for 300 epochs. After the compressor is trained, we run separately the encoder part of the compressor on the remaining training data as well as the test set to generate the new dataset which is used to train the reduced CNN later.
– **Simplified Neural Network for Recognition Task**
By using the encoder we can compress the input image to a smaller one. We have tried to compress the original image to different size and train with CNN. As for the CNN, we remove the two pooling layers in the original implementation to compare the effect of pre-training and in-training compression. The rest settings of CNN remains the same as the original implementation we describe in Section 2.2.

3 Results for Spoken Letter Recognition

In this section, we describe the experiments we conduct to determine the performance of the neural networks we designed for spoken letter recognition task. For consistency and convenience, we refer to the original neural network for spoken letter recognition task as *net-671* and to the simplified neural network as *net-64*. The neural network we use to compress the feature vectors will be referred to as *compressor*.

3.1 Experimental Setup

We implement the three neural networks discussed above, namely *net-671*,*net-64* and *compressor*, in PyTorch. All our experiments, including training, compressing and testing are run on a NVIDIA GTX 1060 GPU. We will use the training time of neural networks as the measure of computational performance.

Dataset The dataset we use to train and test our networks is UCI's ISOLET [5] dataset for spoken English letter recognition task. The dataset contains 7797 instances from 150 subjects who spoke the name of each letter in the English alphabet twice. The subjects are grouped into 5 groups of 30 speakers. Four groups out of five are used as training set and the other group is used for testing. As we mentioned in Section 2.4, we need separate data to train our *compressor*. Therefore, we split the original training data (data of 4 groups) into two parts: 20% of the training data are used to train the *compressor* and the rest 80% of data are used as normal training set to train *net-671* and *net-64*. Each instance in the original dataset contains 617 features. The features include statistics of the speakers' voice records, including spectral coefficients, contour features, etc. However, the exact order of appearance of the features is not known. Other detailed information can be found at UCI's website [1].

Performance Measure We collect two types of statistics during the experiments to measure the performance of our neural networks. First, we collect the prediction accuracy of *net-671* and *net-64* on test set after each training epoch. Then we also measure the total time used in training the networks. For *net-671*, we collect the total time spent in 50 training epochs and for *net-64*, we collect the total time spend in training *net-64* and the training of *compressor*.

3.2 Experimental Results

We now show our experimental results on test set accuracy and average cross entropy loss after each training epoch and the training time. Figure 1 shows the average cross entropy loss (left axis) and prediction accuracy (right axis) after each training epoch for *net-671* and *net-64*. As shown in figure, both *net-671* (93.59%) and *net-64* (94.23%) achieve high accuracy of over 90% on the test and *net-64* gain a slightly higher scores on accuracy. Figure 1 also shows that *net-64* converges faster then *net-671*. This experimental result indicates that our reduction method applied to simplify the neural network for the spoke letter recognition task will slightly improve the convergence speed and the accuracy on test set.

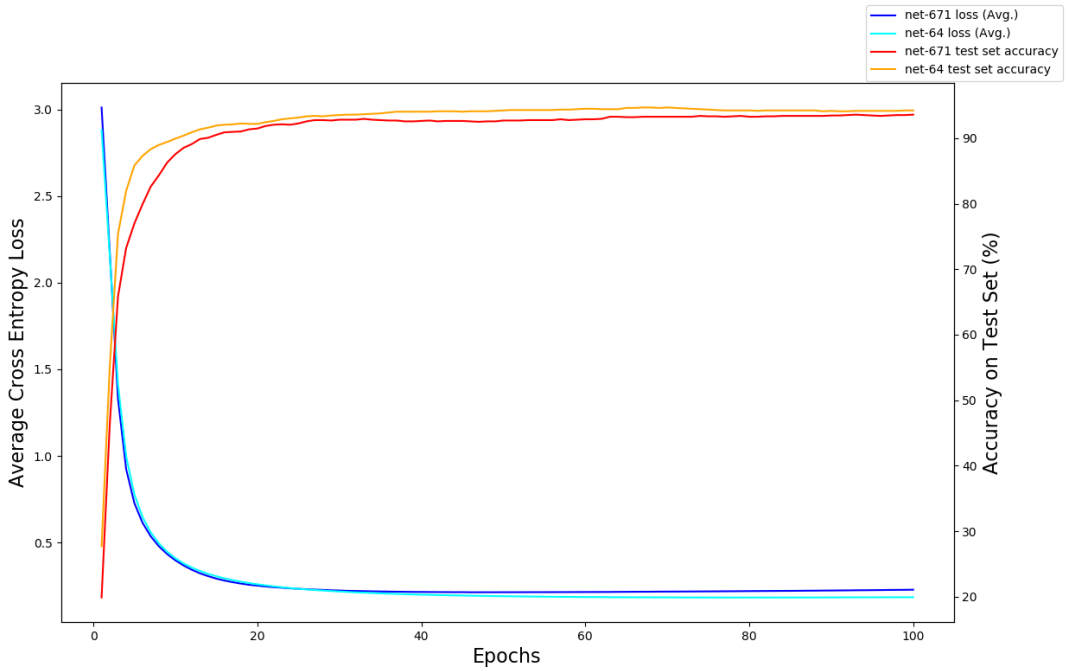

Fig. 1. Test set accuracy and loss for *net671* and *64*

We also report that the we observe an significant improvement in computational performance. The training time of the recognition network is reduced by 20.95% (15.36 seconds for *net-64* and 19.43 seconds for *net-671*). We note that for the measure of training time, we also include the time for compressing the original feature vectors in the training time of *net-64*. This suggests that our reduction approach can significantly improve the computational performance and slightly improve the test set accuracy.

Accuracy Comparison to Other Work The neural network we designed works well on the ISOLET dataset. We achieve an accuracy of 94.23% with *net-64* on test set. Other works on this recognition task with the same ISOLET dataset reports a slightly better accuracy of 95.7% using a neural network with much more units (695) than *net-64* (112). This shows that our reduction achieve a good performance while significantly improves the training efficiency.

4 Results for Handwritten Letter Recognition

In this section, we describe the experiments we conduct to determine the performance of the CNN for handwritten letter recognition. For consistency and convenience, we refer to the original CNN as *cnn-pooling* and to the CNNs without pooling layers whose input image has width of n as *cnn-n*. For example, if we compress the input images to size $14 * 14$, we will refer to the CNN used to classify the images as *cnn-14*. The neural network we use to compress the image will be referred to as *encoder*.

4.1 Experimental Setup

We implement in PyTorch the neural networks *cnn-pooling*, *encoder* and *cnn-n* for $n = 24, 22, 20, 18, 16, 14$. As the previous experiment, all our experiments are run on a NVIDIA GTX 1060 GPU. We will use the training time of neural networks as the measure of computational performance and the accuracy and average loss on test set as performance measures.

Dataset We use EMNIST letter dataset [4] to train and test our model. The dataset contains 26 classes and 145,600 samples. 124,800 of them are used as training data and the rest 20,800 are testing samples. The data samples are distributed evenly over all classes. The label of each data sample is a letter and the features of the data sample is an image of size $28 * 28$. We set the mini-batch size to 100, and therefore there would be 1248 mini-batches for training dataset in total. Other details of the dataset can be found in [4].

Performance Measure We use the same performance measure as the spoken letter recognition task. We collect the prediction accuracy on test set after each training epoch as well as the training time of *cnn-pooling* and of all *cnn-n*s. We also collect the training time of *encoder* network.

4.2 Experimental Results

We now show our experimental results for handwritten letter recognition task. We describe our results in terms of test set accuracy and the training time of networks to examine the effectiveness of our networks designed for letter recognition and input compression respectively.

Accuracy on Test Set Figure 2 shows the average entropy loss (left axis) and test set accuracy (right axis) after each training epoch for `cnn-pooling` and *cnn-14*. These two networks has the same number of parameters (664,923 for both). The difference is that *cnn-pooling* performs average pooling after each convolution layer while *cnn-14* uses no pooling technique. As shown in figure, both *cnn-pooling* and *cnn-14* achieves high accuracy of over 90% on test set. However, *cnn-pooling* performs slightly better than *cnn-14* on this dataset. The same task is accomplished in original EMNIST work paper by using a fully connected neural network, and the authors report a test accuracy of 85.15%. This also shows the advantages of CNN in retrieving information from images.

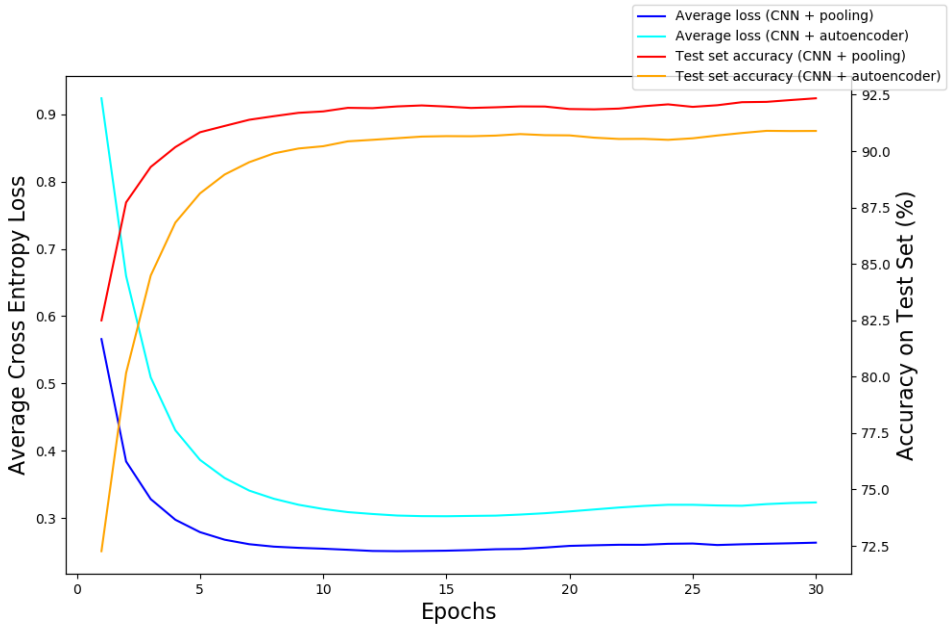

Fig. 2. Performance of *cnn-pooling* and *cnn-14*

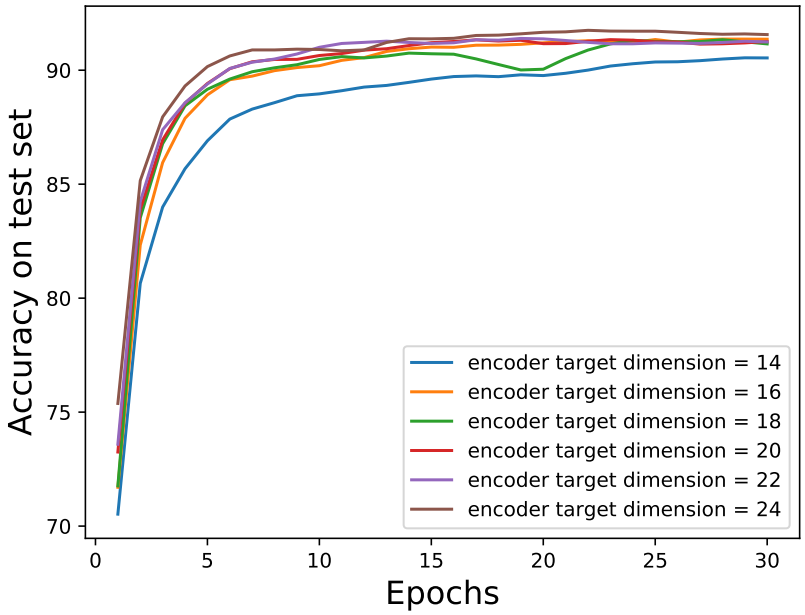

Fig. 3. Test set accuracy using CNN without pooling when compressing the image input into different size.

We also train different *encoders* to compress the input images and then use the compressed images to train and test the CNN with reduced input size. Specifically, the original image $(28 * 28)$ are compressed to $n * n$ where $n = 24, 22, 20, 18, 16, 14$. Figure 3 shows the prediction accuracy on test set for different compressed image input size. As we can see in the figure, all the *cnn-n* achieves a test accuracy over 90%, which is quite close to the result we got from *cnn-pooling*.

Training Time and Size of Network In the history of CNN, the application of the technique is always constrained by computational resources [10]. Though techniques such pooling are used to reduce the requirement of computational resources, training a CNN can still be very computational expensive. One reason for this computational expensiveness is that the size of normal representation of images increases at n^2 speed while the width of image n increases. As we have demonstrated in Section 3, the input compression technique can significantly reduce the training time of neural networks, and we now describe our experimental results for training time for handwritten letter recognition tasks using CNNs with different size of inputs.

Table 1. Number of parameters of different CNN

Input image width	Using pooling	24	22	20	18	16	14
# Parameters in CNN	664,923	19,015,003	13,247,835	8,529,243	4,859,227	2,237,787	664,923

We compare the number of parameters in CNN when using pooling technique and when compressing the input images to a certain width. The numbers of parameters of CNNs are shown in Table 1. As we can see in the table, two layers of pooling reduces the number of parameters to the same number of parameter in *cnn-14*. Figure 4 shows the training time of each CNN. The red line indicates the training time of *cnn-pooling* while the blue bar suggests the training time of each *encoder* and the orange bar shows the training time of each *cnn-n*. As we can see in the figure, the training time of *cnn-14* (44.91 seconds for CNN, 19.14 seconds for encoder, 64.05 seconds in total) is significantly smaller (reduced by 30.40%) than the training time of *cnn-pooling* (92.03 seconds) even though they have the same number of parameters. This suggests that compressing the input image can achieve the same result as pooling in reducing the size of network, and compressing can save more training time than pooling. The reason for saving more training time can be that pooling needs more logical operations, which can break the parallelism. This

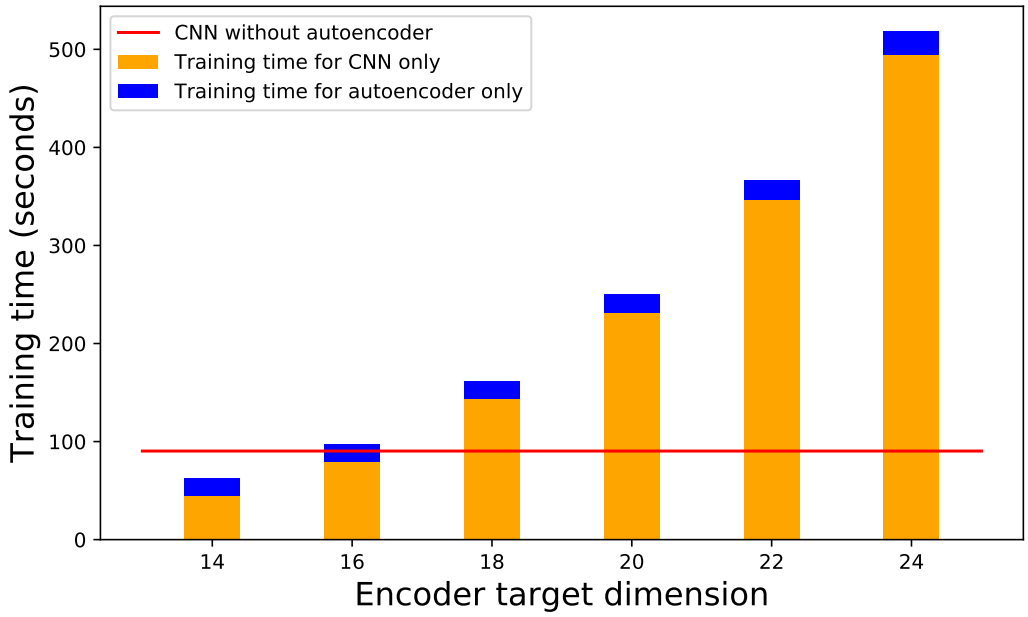

Fig. 4. Training time of CNN without pooling when compressing the image input into different size.

can significantly slow down the training speed, especially on devices like GPUs. We also note that if we compress the input images to $16 * 16$, the *cnn-16* will have $2,237,787$ parameters, but the training time is quite close to that of *cnn-pooling*. By comparing the test accuracy, we find that *cnn-16* achieves an accuracy of 91.36%, which is close to the accuracy of 92.34% by *cnn-pooling*. This results shows a shortcoming of our input compression technique: though the input compression can significantly reduce the training time of CNNs, the compression may be lack of locality, which is essential for extracting information from images, and therefore the accuracy of neural networks may decrease due to the lost or indistinguishable information.

5 Discussion

Neural network is a powerful machine learning technique to resolve the real word problems. The structure of a neural networks can affect the performance and computational cost significantly and the features we used as input to the neural networks can also have a significant impact on the performance of network. Our experiments shows that though neural networks can be resistant to the redundant information contained in the input features, the high dimensional feature vectors will introduce additional complexity to the structure of neural networks, which may further cause computational issues. The computational overhead introduced in this case is generally hard to identify because the performances on other aspects, for example prediction accuracy, may not be impacted significantly. From information theory perspective, the encoder part of our *compressor* or *encoder* removes the redundant information in features of the dataset and therefore increases the entropy of the training data, which makes the training more efficient. However, as we discussed in Section 4.2, a naive compression method may make the information in original data lose or become indistinguishable.

6 Conclusion and Future Work

In this paper use neural networks to resolve two letter recognition tasks: spoken English letter recognition and handwritten letter recognition. For spoken letter recognition, we design and implement a fully connected feed-forward neural network to resolve the English spoken letter recognition task. We conduct extensive experiments to measure the performance of neural network and other techniques we introduce and implement. The experimental results shows that this neural network achieves a test set accuracy of 93.59%. We then apply a feature compression on the dataset to reduce the dimension of input feature vectors. Our *compressor* reduces the dimension of input

feature vectors by 90.5%. This compression technique slightly improves the test set accuracy from 93.59% to 94.23% and reduce the training time of the network by 20.95%. We also compare our reduced neural network *net-64* with the one solving the same problem on the same dataset [6]. The experiments shows that *net-64* can achieve a similar accuracy but is much smaller in size and also should have a better performance in training time.

For handwritten letter recognition task, we design and implement convolution neural networks to solve the problem. We also apply the input compression technique to reduce training time and network size, and conductive extensive experiments to examine the effectiveness of our techniques. We report a test set accuracy of 92.34% on *cnn-pooling* on EMNIST letter dataset while the authors of the dataset reports an accuracy of 85.15% using a fully connected neural network. We train and test different *cnn-n* networks for $n = 24, 22, 20, 18, 16, 14$. From the experimental results we find that our technique can reduce the training time by 30.40% by compressing the input and reduce the size of CNN to the size of *cnn-pooling* while only make the accuracy slightly decrease.

From a information theory perspective, our *compressor* removes the redundant information contained in dataset and increase the entropy of the training data. A reasonable compression of input features can help with computational issues while keep the performance at a similar level. Our experimental results also support this. However, the *compressor* is still designed by human with the knowledge from dataset. An interest future work can be if there exists a theoretical boundary of such compression and whether we can learning from the data automatically and build an adaptive approach to design such a compressor.

References

1. Isolet data set (1994), https://archive.ics.uci.edu/ml/datasets/isolet
2. Ciresan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Convolutional neural network committees for handwritten character classification. In: Document Analysis and Recognition (ICDAR), 2011 International Conference on. pp. 1135–1139. IEEE (2011)
3. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289 (2015)
4. Cohen, G., Afshar, S., Tapson, J., van Schaik, A.: Emnist: an extension of mnist to handwritten letters. arXiv preprint arXiv:1702.05373 (2017)
5. Dheeru, D., Karra Taniskidou, E.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
6. Fanty, M., Cole, R.: Spoken letter recognition. In: Advances in Neural Information Processing Systems. pp. 220–226 (1991)
7. Ganis, M., Wilson, C.L., Blue, J.L.: Neural network-based systems for handprint ocr applications. IEEE Transactions on Image Processing **7**(8), 1097–1112 (1998)
8. Gedeon, T., Harris, D.: Progressive image compression. In: Neural Networks, 1992. IJCNN., International Joint Conference on. vol. 4, pp. 403–407. IEEE (1992)
9. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
10. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
11. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proc. icml. vol. 30, p. 3 (2013)
12. Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., Khudanpur, S.: Recurrent neural network based language model. In: Eleventh Annual Conference of the International Speech Communication Association (2010)
13. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10). pp. 807–814 (2010)
14. Rajavelu, A., Musavi, M.T., Shirvaikar, M.V.: A neural network approach to character recognition. Neural networks **2**(5), 387–393 (1989)
15. Shamsher, I., Ahmad, Z., Orakzai, J.K., Adnan, A.: Ocr for printed urdu script using feed forward neural network. In: Proceedings of World Academy of Science, Engineering and Technology. vol. 23, pp. 172–175 (2007)
16. Sharma, A., Imoto, S., Miyano, S.: A top-r feature selection algorithm for microarray gene expression data. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) **9**(3), 754–764 (2012)
17. Singh, R., Yadav, C., Verma, P., Yadav, V.: Optical character recognition (ocr) for printed devnagari script using artificial neural network. International Journal of Computer Science & Communication **1**(1), 91–95 (2010)
18. Xie, J., Wang, C.: Using support vector machines with a novel hybrid feature selection method for diagnosis of erythemato-squamous diseases. Expert Systems with Applications **38**(5), 5809–5815 (2011)