Image Compression Using Three Auto-Associative Networks

Ao Feng [1]
Research School of Computer Science,
Australian National University,

Canberra, ACT,2601
u5962333@anu.edu.au

Abstract. In this paper, we consider Auto-associative Networks [1] as an image compression method. We created three Networks, Convolutional Neural Networks(CNN) Auto-encoder, Auto-associative Neural Networks(ANN) and Auto-associative Neural Networks using shared weight. We would use these methods to do image compression on the same dataset, and recover the image from compressed image. In order to show the quality of image compression, we would put recovered image in a multi-classification classifier using Convolutional Neural Networks. To analyze three methods, we compare three methods' performance, the result shows the CNN Auto-encoder get the best accuracy and the shared weight Auto-associative Network spend less time for training than other methods.

Keywords: Auto-Associative Networks, Image Compression, Convolutional Neural Networks, Shared Weight

1 Introduction

Image compression is method to compress the data of digital images. This method is widely used in machine learning and deep learning field. Image compression can keep the important feature of the original picture and reduce the store space. This is a good way to store big data because it can save space. However, sometimes the quality of decompressed image is very bad, it can't keep the important feature of the image. In order to find a good method to compress image not only in good quality but also trained fast, I decided to test some method for image compression.

There are many methods for image compression. In machine learning field, Principal component analysis(PCA) [2] is a very famous dimension reduction method. Using PCA can reduce the number of features, which means the compressed data will only contain the most important feature of the image. Linear Discriminant Analysis(LDA) [3], is also a widely used method, this method is to find linear combination of features that can best separate the classes of objects. Sparse autoencoder[3] is another famous method, which is an approach to automatically learn features from unlabeled data, it can be widely used in unsupervised learning fields. These three methods are not perfect, they all have limitation, for example, PCA might lose important features, LDA is not suitable for data which is not belong to Gaussian distribution.

In this paper, we use a Convolutional Neural Networks to do classification for the image. We use three methods to do image compression, and we recover the compressed data and use them as input for the CNN classifier. The result shows the decoded image using three methods can keep most features of the original image. The result also shows that the neural network using shared weight technique [4], which is proposed by Tom can reduce the training time of image compression model.

2 Method

The flow-chart below (Fig.1) shows the implementation of the system. The system contains three steps, it will process the dataset, use three Auto-Associative methods to compress image, recover the image and use CNN classifier do classification.

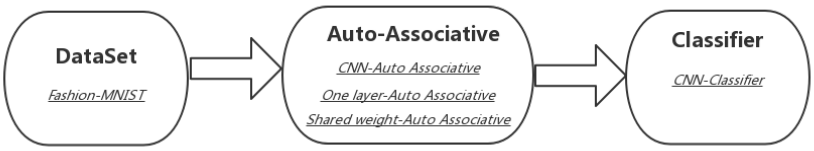

Fig. 1. System Structure

2.1 Data Set

The data set used for this system is Fashion-MNIST. Fashion-MNIST is a dataset comprising of 28 × 28 grayscale images of 70,000 fashion products from 10 categories, the size of the whole data set contains 70,000 images [5]. The training set has 60,000 fashion product images, the test set has 10,000 fashion product images. For the dataset, it has 10 categories, which are T-shirt/Top, Trouser, Pullover, Dress, Coat, Sandals, Shirt, Sneaker, Bag and Ankle boots.

Table 1. Files contained in the Fashion-MNIST dataset[5].

Name	Description	#Example	Size
train-images-idx3-ubyte.gz	Training set images	60,000	25MBytes
train-labels-idx1-ubyte.gz	Training set labels	60,000	140Bytes
t10k-images-idx3-ubyte.gz	Test set images	10,000	4.2MBytes
t10k-labels-idx1-ubyte.gz	Test set labels	10,000	92Bytes

Fashion-MNIST is very similarity with another dataset, MNIST, which is a very famous dataset for machine learning and deep learning. However, MNIST sometimes perform too well, it is hard to show the real performance of the model. People can easily get a good result using MNIST. This's the reason why I choose Fashion-MNIST rather than MNIST. Another advantage of Fashion-MNIST is that the size of the dataset is enough for the model. The whole dataset has 70,000 examples, that is enough for training and testing. For every single image, the size is 28*28, that's not a very big image, so it won't spend too much time on training. According to these reasons, I decide to use Fashion-MNIST as dataset for this system.

In order to have a better visual experience, I change the data file into image with jpg format. People can check every example of the dataset, and see what it looks like. In next steps, when the system need to load the data, it will directly load the jpg image. The dataset is different for different models, for the Auto-Associative model, the size of training data is 5,000, the size of test data is 2,000. For the CNN-Classifier, it uses the whole dataset, that is 60,000 for training and 10,000 for testing.

2.2 Auto-Associative Networks

2.2.1 CNN-Auto-encoder Networks

In this model, we build autoencoder network using CNN. The input of the model is the image we want to compress, the output is the image recovered form compressed image. It contains encoder step and decoder step, in encoder step, the image will be compressed, the compressed data will be stored after encoder step. The decoder step would recover the compressed data, and output the recovered image.

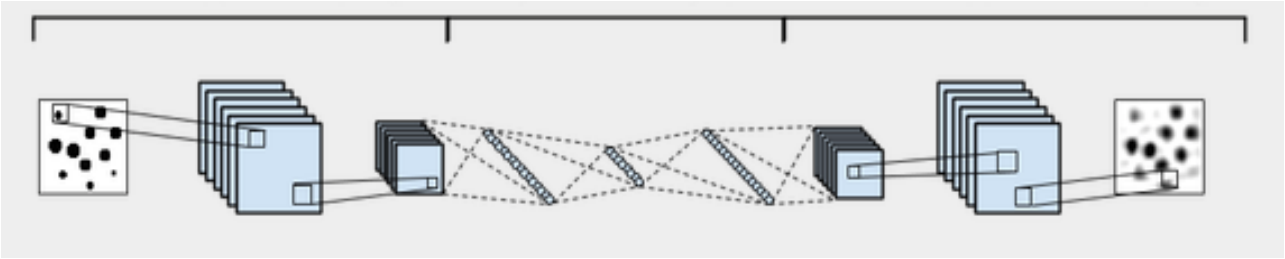

Fig. 2. CNN-Auto-associative network topology [7]

Fig2 shows the progress for a normal CNN autoencoder. In encoder step, there is a convolutional layer, it uses filters to scan the input, the pooling layer will do pooling operation after all filters have scan the input. After convolution step, the input image has been compressed. The data will then go though a fully connected layer and get the final encoder data. The next step is very similarity with convolution step, but in a reversed order. This step is Deconvolutional step, it contains unpooling step and deconvolutional step. Fig3 shows how to complete de-convolution step and un-pooling step. For de-convolution step, every kernel has its kernel size, stride and padding. De-convolution step will base on the value of the current layer and use these value to recompute the value of the original input. For example, the kernel size is 3*3, for every single element in the filter layer, it was calculated form a 3*3 kernel of the input. We can recover the value of a 3*3 kernel of the input according to the value in the filter layer. Once we calculated all the value of the filter layer, we can recover the value of the whole input layer. Un-pooing step is much easier to understand, we just use the value in pooling layer to fill in the filter layer. Because all the pooling value is got from the kernel we set before, now we just let the value in the kernel all equal to the pooling value.

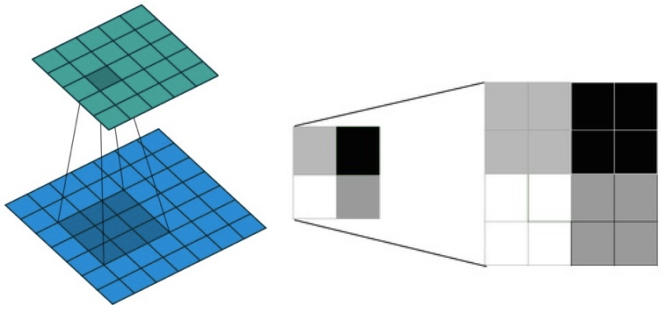

Fig. 3. De-convolution and un-pooling step [7][8]

The model in this paper is similarity with Fig2 shows. But it doesn't have full connect layer. It only has convolutional layer and de-convolutional layer. The model has two convolutional layer. The first convolutional layer uses 25 kernel with size 3*3. The second convolutional layer uses 10 kernel with size 3*3. After every convolutional layer, they are both going through a 2*2 max pooling layer. These two convolutional layers can compress the image. The recover step uses two de-convolutional layers, one layer uses 25 kernel another layer uses 10 kernel. There is no un-pooling step during the de-convolutional step. In this model, the size of input image is 28*28, and after the convolutional step, it was compressed into size of 2*2. The de-convolutional step recovered the image for 2*2 into 28*28. To train the model, the model use 5,000 sample as training set and 2,000 sample as test set. The whole training progress contains 200 epochs, using mean squared error as lost function and Adam for the optimizer.

2.2.2 Auto-Associative Network

This model is a feed back neural network with three layers [6]. The model has three layers, input layer, hidden layer and output layer. Three layers are fullly connected. Like fig4 shows, the number of neurons in input layer is equal with the number of neurons in output layer. The input will go through active function, in this model we use sigmoid function, and go to the hidden layer. The data in hidden layer is the compressed data, these data will flow through to the output layer as output of the model. The model in this paper, the number of input neuron is 784, since the size of the image is 28*28, the number of neurons in output layer is also 784. The input data will be stored in 400 hidden neurons. In order to minimize the loss between input and output, we use mean squared errors as lost function and Stochastic Gradient Descent for the optimizer. The model has been trained for 200 epochs.

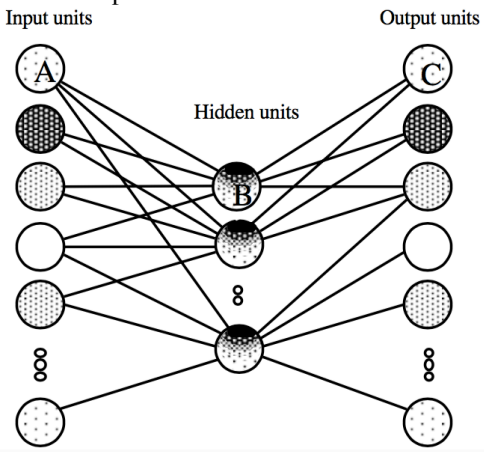

Fig. 4. Auto-Associative Networks topology [6]

2.2.3 Auto-Associative Network using shared weight

This model is very similar with the model in sec 2.2.2, but it uses the shared weight technique, which is proposed by Tom [9]. In this model, the weights form input layer to hidden layer is equal with the weights form hidden layer to output layer. Like the fig5 shows, the model has 256 input neurons, the weights from A to B is equal with the weights from B to C. For example, x1 neuron in input layer, it connects to all the neurons in hidden layer, represent these weights by number 1, we can see from fig5, the weights of hidden neurons connected to y1 neurons in output layer is also marked with number 1. This means the weights form x1 to hidden layer is the same with the weights from hidden layer to y1. This is a good way to improve the network, because it will only update the weights once for every training time rather twice, this will reduce the training time when the network is very complex. Another advantage is using shared weight can avoid over fitting problem. Because it can reduce the freedom degree of the parameters.

The model in this paper is nearly the same with the model in sec 2.2.2, it also has 784 input neurons, 400 hidden neurons and 784 output neurons. The loss function and optimizer are also the same. The different place is in this model, I forced the weights form input layer to hidden layer equal with the weights from hidden layer to output layer.

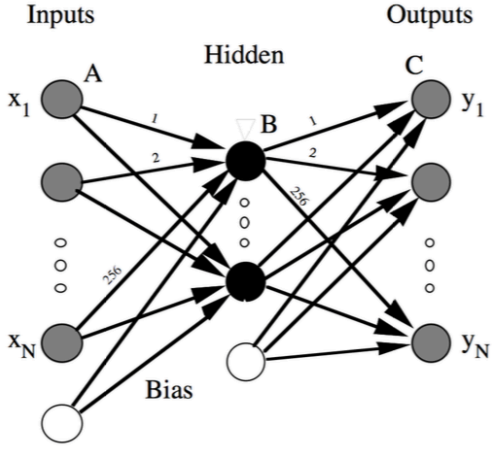

Fig. 5. Auto-Associative Networks with shared weights topology [9]

2.3 CNN Classifier

In the previous steps, I trained three image compression models. In order to test their performance on Fashion-MNIST classification problem, I build this CNN model to do classification. This model can make classification for the input fashion product image. The structure of the model shown as Fig6. The input is a sneaker image with size of 28*28. It has two convolution layers and two pooling layers. The first convolution layer uses 10 filters with 5*5 kernel. The size of the filter is 24*24. After pooling layer, the size becomes 12*12. The second convolution layer use 20 filters with 5*5 kernel. The size becomes 8*8. After the second pooling layer, the size becomes 4*4. The 20 filters will all be put in the fully connected layer. The output layer would give the classification for the input image. The model is trained for 20 epochs with 60,000 examples.

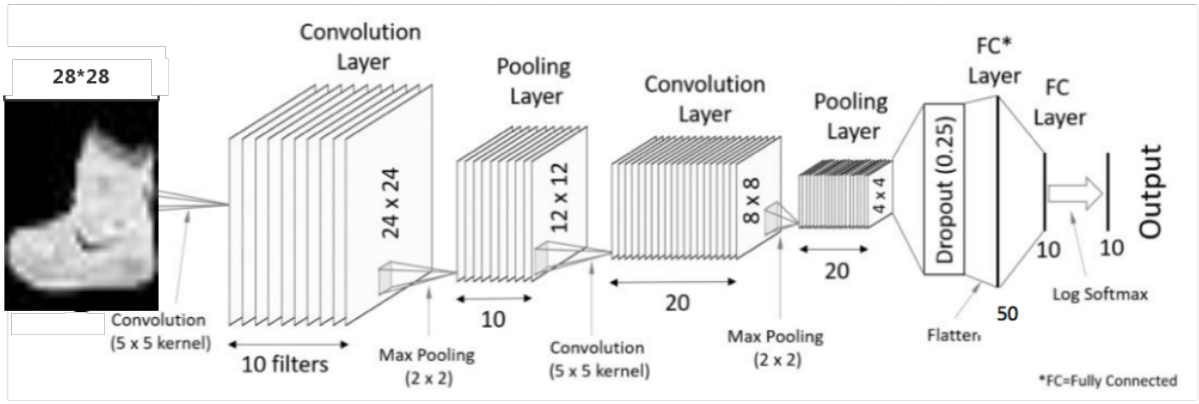

Fig. 6. Convolutional Neural Network Classifier Structure

3 Result and Discussion

As we discussed in section 2, we build three methods to compress image. Fig7 show the training progress of three methods. The CNN-Autoencoder perform best among the three methods, and it has the lowest loss after 200 epochs training. The Auto-Associative network performs medium, it between shared weight method and CNN method. The shared weight has the highest loss. According to the result, all the methods have low loss. It indicates three methods are all well trained.

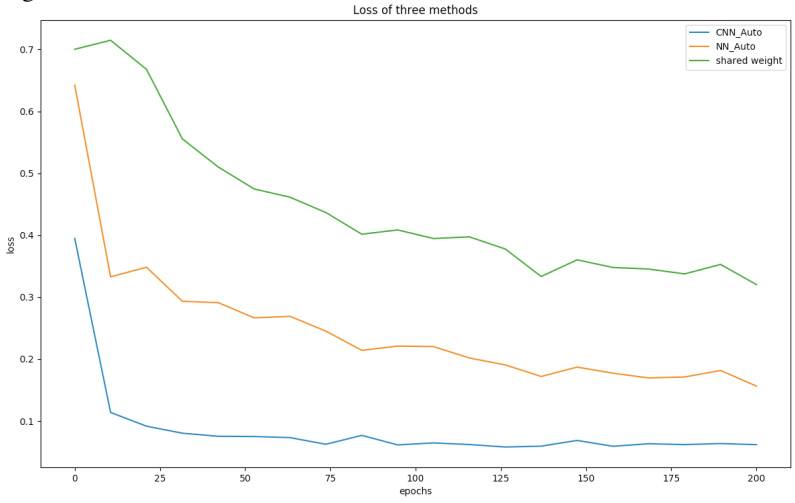

Fig. 7. The loss of three image compression methods

In order to have a better visual experience, we print some compressed images during training progress. We show 64 compressed images one epoch, we show the image on 1th epoch, 100th epoch and 200th epoch. In fig8, the first row is the result using CNN-method, the second row is using the normal Auto-Associative network, and the third row using shared weight technique. In the first column of fig8, we can see all the images look not much clear. However, in the second column of fig8, which means the result of 100th epoch, CNN method can show the picture clearly, the Auto-Associative network can show the shape of the picture but not very clear. Finally, in the last column of fig8, CNN method can show the result very clearly, ANN method can show the shape of the image with a little unclear. However, the result using shared weight method can't see it clearly, even after 200 epoch training, it still looks very unclear. Compared with its first training epoch, we can see it has improved a lot.

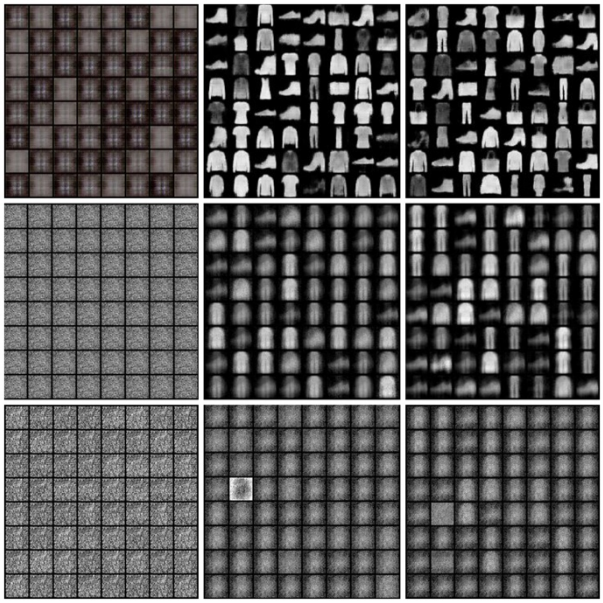

Fig. 8. Compressed Images using three method

To test the quality of compressed image, we would use a CNN classifier to test the compressed image. For the CNN classifier, table2 show performance of the classifier during 20 epoch of training. After training with 60,000 examples, the final accuracy on test set in 85%. We use compressed images as input, and record the result. The result shown on the table3. In general, all three methods have a good result(accuracy>70%). The CNN method performed best among three methods, it has lowest final loss and highest accuracy. For a normal ANN and ANN with shared weight, they both performed worse than CNN method. But if we only compare ANN method and ANN with shared weight, they have very close accuracy, but shared weight technique can save training time. If the data size is very large, shared weight can save much training time with a little accuracy loss.

Table 2. CNN Classifier Training Progress

epoch Number	Training loss	Test set Accuracy
1	0.852438	71%
5	0.606686	79%
10	0.509882	82%
15	0.422064	84%
20	**0.431602**	**85%**

Table 3. Result of three methods on CNN-classifier

Method Name	Final loss	Accuracy	Training Time /s
CNN	**0.4968**	**81%**	622.595228
ANN	0.7037	73%	597.469302
ANN(Shared weight)	0.8070	70%	**584.928798**

There are also other people do similarity research. Tyler did some research on PCA and auto-encoder, he uses these two methods as a way for dimension reduction [10]. According to Tyler's result, the auto-encoder perform much better than PCA. Tyler uses two confusion matrix to show the result of PCA and auto-encoder. From the confusion matrix of PCA [10], there are some label only has 23% accuracy or 48% accuracy. From the confusion matrix of auto-encoder, it has a good result. Most labels have accuracy greater than 90%, only two labels have low accuracy (72% ,65%). Compare Tyler's result with our result. Our result is much better than PCA. It can be seen that PCA sometimes may lose important features during image compression. But the auto-encoder from Tyler's article has a very high accuracy, it performs even better than CNN method in our model. The reason might be the number of hidden neurons. If the size is very small, it may not contain all the important feature. Another reason might be the design of our model. For example, the CNN method might use one more convolutional layer, and the ANN model can try to use other learning rate or optimize method. Also, more training epoch may also influence the accuracy.

4 Conclusion and Future Work

4.1 Conclusion

To find a good method for image compression, we use three methods to compress the image and compare the result with each other. The CNN auto-encoder method is a good method for dealing with image problem, but some times it needs

long time for training. The auto-associative network is a not bad method, its structure is simple, and won't spend too much time on training, it can also get a good result on image compression. For the auto-associative network using shared weight technique, which is proposed by Tom [9], it is the fastest model among tree model in this paper, it can train the model with less time. However, its disadvantage is also obvious, sometimes it can't keep the quality of the original image. In conclusion, all the methods we used in this paper can do image compression, the result shows they can keep most feature of the original images. The CNN auto-encoder network can compress the image with best quality, and the shared weight technique can save the training time.

4.2 Future Work

Though the three models get a good result on image compression, there are still a lot work to do in the future. First, determine a suitable structure for the model is important. For example, the number of convolution layer in the CNN autoencoder model, the number of hidden neurons in ANN model. Second, we can add some noise on the image and check whether the model can filter the noise. We can compare the original image (add noise) with recovered image from compression image. If the model can filter the nose of the original image, the network can not only compress the image but also be a good way for data processing. Third, since the CNN autoencoder model spend too much time on training, we can find a way to increase the speed of training. For example, we can use pruning technique in the fully connect layer or we can try to quantize the fully connected layer using product quantization [11]. Finally, there are still many other methods to compress image, such as Deep Autoencoder Networks, ISOMap and PCA. We can explore these methods and test their performance in the future.

References

[1]Gedeon,T.D.,Harris,D.:Progressiveimagecompression.InNeuralNetworks,1992. IJCNN., International Joint Conference on (Vol. 4, pp. 403-407). IEEE (1992).

[2]Jolliffe, Ian T. "Principal component analysis and factor analysis." Principal component analysis.pp.115-128, Springer, New York, NY, 1986.

[3]Izenman, Alan Julian. "Linear discriminant analysis." Modern multivariate statistical techniques. pp.237-280,Springer, New York, NY.

[4]A. Ng, "Sparse autoencoder. "CS294A Lecture Notes, 2011.

[5]Xiao, Han, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." arXiv preprint arXiv:1708.07747 (2017).

[6]Gedeon, T. D., J. A. Catalan, and J. Jin. "Image Compression using Shared Weights and Bidirectional Networks." Proceedings 2nd International ICSC Symposium on Soft Computing (SOCO'97).

[7]M. Swarbrick Jones, "Convolutional autoencoders in python/theano/lasagne", Mike Swarbrick Jones' Blog, 2018. [Online]. Available: https://swarbrickjones.wordpress.com/2015/04/29/convolutional-autoencoders-in-pythontheanolasagne/. [Accessed: 28- May- 2018].

[8]J. Li, "One by One [1 x 1] Convolution - counter-intuitively useful", Frank, 2018. [Online]. Available: http://www.notehub.cn/2016/10/15/algo/ml/OnebyOne_Convolution/. [Accessed: 28- May- 2018].6. National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov

[9]Gedeon, T. D. "Stochastic bidirectional training." Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on. Vol. 2. IEEE, 1998.

[10]Manning-Dahan, Tyler. "PCA and Autoencoders." (2018).

[11]Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149 (2015).