Comparison between Classification with/without Pruning and Generative Modeling with U-Net

Jiulin Zong[1]

[1] Research School of Computer Science, Australian National University, Australia
u6921252@anu.edu.au

Abstract. With the development of deep learning and computer hardware, artificial intelligence is gradually moving from academic to industrial. There are more and more data sets and more parameters. Although it improves the accuracy and generalization of the model to a certain extent, it is inevitable that the parameters of the model become more and more and more data, the demand for computer hardware and training time is also increasing, but according to the results of the paper, there is a great degree of redundancy between the active units of neural network. In the last paper I used autoencoder to compress EEG data, and then uses the encoder trained by autoencoder to compress the original data, and then trains a three-layer neural network classifier through the compressed data, finally realizes the pruning and classification. In this paper I will try to apply generative modelling and U-net and try to compare their performance.

Keywords: Auto Encoder, Pruning, Generative Modelling, U-Net

1 Introduction

With the continuous development of deep learning technology, it is gradually applied to the increasingly complex reality background. In order to deal with complex practical problems, people use more and more deep neural network model (the number of hidden layer is more and more), or more and more large (the number of hidden neurons in a single hidden layer is more and more), which leads to the increase of the amount of calculation, the calculation time and memory occupied by parameters. In order to control the size of the model, people use various methods to minimize the use of parameters, such as reducing the number of hidden layers or the number of hidden neurons in a single hidden layer. However, the reduction of hidden units means that the number of features that the model can extract from the input data is reduced (the function of each hidden neuron can be regarded as extracting an independent feature) [4].

At the beginning, the selection of the appropriate model structure and the number of hidden units often depends on the experience of the designer, and there is no fixed and accurate method for reference. Therefore, people usually design the neural network a little larger to ensure that there are enough or too many parameters to extract the required features, even if this method has the risk of over fitting [4]. In the last project we tried to introduce a network pruning method, which can be used to remove the redundant parameters of the trained deep neural network. However, the performance was not as good as we expected. So in this project we will try to use a combination of generative modelling and U-net, and compare their performances.

As for the dataset and processing, the electroencephalography (EEG) signals is still being used, but in a larger and form which focus on the images [1]. We use a total of 11057 samples of egg data, each sample has 192 dimensional features. The data label is "y_ alcoholic", "y _ alcoholic" is a N-vector of binary labels, 1 if the subject is an alcoholic,0 if the subject is not [2].In the last project, we use two models: 1. Use only three-layer fully connected network; 2. Firstly use autoencoder to encode and compress the data, and then use three-layer fully connected network to classify the encoded and compressed data. Based on the previous model, in this project we will firstly implement a generative model and test its performance. Then adds U-net and compare the results.

2 Method

2.1 Method for Implementing Auto Encoder

In 1986, Rumelhart proposed the concept of automatic encoder and applied it to high-dimensional complex data processing, which promoted the development of neural network. Self coding neural network is an unsupervised learning algorithm, which uses back propagation algorithm and makes the target value equal to the input value. Autoencoder is a kind of neural network, which can be regarded as two parts: an encoder function H = f (x) and a reconstructed decoder r = g (H) [1]. Automatic encoder is an unsupervised neural network model. It can learn the hidden features of input data, which is called coding. At the same time, it can reconstruct the original input data with the new features, which is called decoding. Intuitively, automatic encoder can be used for feature dimension reduction, similar to PCA, but its performance is better than PCA, because neural network model can extract more effective new features. In addition to feature dimensionality reduction, the new features learned by the automatic encoder can be fed into the supervised learning model, so the automatic encoder can act as a feature extractor [5].

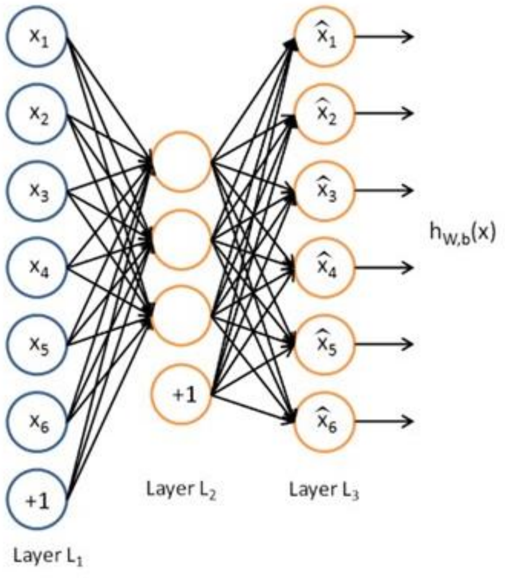


Figure 1. A simple sample model

2.2 Optimization of the Neural Network

We also tried to apply some optimization methods to improve the accuracy

Adam Optimizer

Adam, a one-step stochastic objective function optimization algorithm based on low order moment adaptive estimation. This method is easy to implement, high computational efficiency and less memory occupation, which is very suitable for big data or big parameter problems. This method is also suitable for non-stationary targets and high noise and / or sparse gradient problems. Hyperparameters have intuitive explanations and usually require little adjustment [6].

Adjust Better Hyperparameters and Neural Units

Better model can be obtained by adjusting super parameters. In this experiment, I adjusted the number of iterations, the number of neurons in each layer, and the learning rate. In order to prevent over fitting and under fitting, the loss functions of the models with different super parameters are compared for many times, and finally a group of relatively optimal models are determined.

Sigmoid Function

Sigmoid function is a mathematical function with characteristic "s" curve or sigmoid curve. A common example of the sigmoid function is the logistic function shown in the first figure, which is defined by a formula: $S(x) = 1/(1+e^{-x}) = e^x/(e^x+1) = 1-S(-x)$.

When using this function to do the classification problem, we can not only predict the category, but also get the approximate probability prediction. This is very useful for many tasks that need to use probability to assist decision-making. Logarithmic probability function is an arbitrary order differentiable convex function, it has good mathematical properties, many numerical optimization algorithms can be directly used to find the optimal solution [7].

The advantage of sigmoid is that the output range is limited, so the data is not easy to diverge in the process of transmission. Of course, there is a corresponding disadvantage, that is, the gradient is too small at saturation. Another advantage of sigmoid is that the output range is (0, 1), so it can be used as the output layer, and the output represents the probability [7]. These are also the reasons why I finally chose sigmoid function as activation function.

2.3 Indicators of Hidden Neuron Functionality

With the continuous development of deep learning technology, it is gradually applied to the increasingly complex reality background. In order to deal with complex practical problems, the neural network model used by people is becoming deeper and deeper, which leads to the increase of calculation amount, calculation time and memory occupied by parameters. In order to control the size of the model, people use various methods to reduce the use of parameters as much as possible, such as reducing the number of hidden layers or the number of hidden neurons in a single hidden layer, but the reduction of hidden units means reducing the number of features that the model may extract from the input data. We continue to optimize the classifier of the model. According to the paper [3], the trained neural network often has a large number of parameter redundancy. The same hidden neuron will extract the same features in different samples, and different hidden units should extract different features for the same input sample. Vector angle analysis is to analyze the similarity of output results of different hidden neurons in the same input sample. Two similar columns of data show that the corresponding two hidden neurons extract similar features, and their functions are similar, so they can be combined; Two opposite columns of data show that the results of the corresponding two hidden neurons counteract each other, so they can be deleted at the same time. We refer to the significance of each hidden unit to determine which hidden layer neurons should be retained in the model.

We use vector angle analysis to judge the significance of hidden units. The output results of multiple input samples in the hidden layer can form a matrix. The neural network designed in this paper uses sigmoid as the activation function in the hidden layer, and the output value of the activation function is in the range of 0 ~ 1. In order to facilitate the later vector angle analysis, we add an offset of - 5 to the output result, so that the range becomes - 0.5 ~ 0.5. Then the vector angle of pairwise neural units is calculated by vector angle formula. After calculating the vector angle, we do the following processing: 1. Two vectors whose vector angle is less than 15 degrees have similar functions, that is to say, they satisfy the linear correlation. We delete one hidden unit and add its parameters (weight and bias) to the other hidden unit; 2. When the angle of two vectors is larger than 165 degrees, we think that their functions are mutually exclusive, and they cancel each other in the calculation process, and delete the two hidden neurons directly.

The calculation formula of vector angle is: angle(u, v) = arccos (u · v / (‖ u · V ‖)). According to the definition of arccosine function, the output range is 0 ~ 180 degrees. Since the output range of sigmoid is 0 ~ 1, the vector angle between two vectors with integer values must be between 0 ~ 90 degrees. In order to get the result of 90 ~ 180 degrees, we add an offset of - 5 to the result of sigmoid activation function, so that a negative value will appear, and the calculated vector angle will cover the range of 90 ~ 180 degrees.

After calculating the vector angle, we do the following processing: 1. Two vectors whose vector angle is less than 15 degrees have similar functions, that is to say, they satisfy the linear correlation. We delete one hidden unit and add its parameters (weight and bias) to the other hidden unit; 2. When the angle of two vectors is larger than 165 degrees, we think that their functions are mutually exclusive, and they cancel each other in the calculation process, and delete the two hidden neurons directly [2].

2.4 Generative Modelling & Discriminative Modelling

The joint probability distribution is estimated by the model. It is used to model randomly generated observations, especially when some hidden parameters are given. In machine learning, it can be used to model data directly (using probability density function to model observed sample data), or as an intermediate step to generate conditional probability density function. By using Bayesian rules, the conditional distribution can be obtained from the generated model. If the observed data is completely generated by the generated model, then the parameters of the generated model can be fitted, thus only the data similarity can be increased. However, few data can be obtained completely by the generating model, so the more accurate way is to model the conditional density function directly, that is, to use classification or regression analysis. Here I designed a deconvolutional neural network for generator.

The discriminative model estimates the conditional distribution. Using positive and negative examples and classification tags, we mainly focus on the edge distribution of the discriminant model. The objective function directly corresponds to the classification accuracy. For discriminator I used CNN.

Probability for different model:

Generative model: P (class, context) = P (class|context) * P (context)

Discriminative model: P (class|context)

2.5 Method for Implementing U-Net

U-net is also a semantic segmentation network based on FCN method. In the original FCN network structure, it has been found that the underlying features are very useful for recognizing the edge of objects. Therefore, two network structures, FCN-16s and FCN-8s, which combine the underlying feature information, are proposed [9].

In U-net network, through the way of jump connection, the low-level features and high-level features are directly combined to achieve accurate pixel classification. The combination of features is cascade.

The whole process of u-net is encoder-decoder. At first, the main function of this structure was not segmentation, but image compression and denoising. This idea can also be used in the original image denoising. The method is to add noise artificially to the original image in the training stage, and then put it into the codec. The goal is to restore the original image [8].

3 Results and Discussion

3.1 Result of Training Classification Model

Firstly, we divide the data into training set and test set with the ratio of 8:2. Then we implement a classification model including four layers of full connection to classify the data directly. After training 40 epochs, the accuracy of the test set is 0.86, and the training time is 44s

3.2 Result of Processing by Autoencoder

We firstly implemented an autoencoder to compress the data. The reconstruction error of autoencoder is as follow:

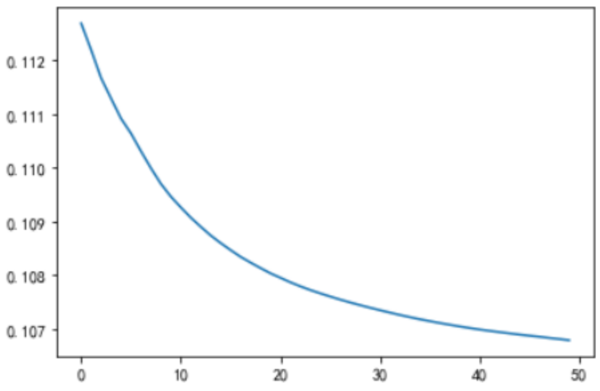


Figure 2. Reconstruction error of autoencoder

Then we use the encoder of autoencoder to compress the original data from 63 * 3 to 16 * 3. Finally, a three-layer fully connected network is used as a classifier to classify the compressed data. The final accuracy is 0.79%

However, the training time of the model is more than 20s, and the memory occupied by the data is also twice as high.

3.3 Result of Pruning

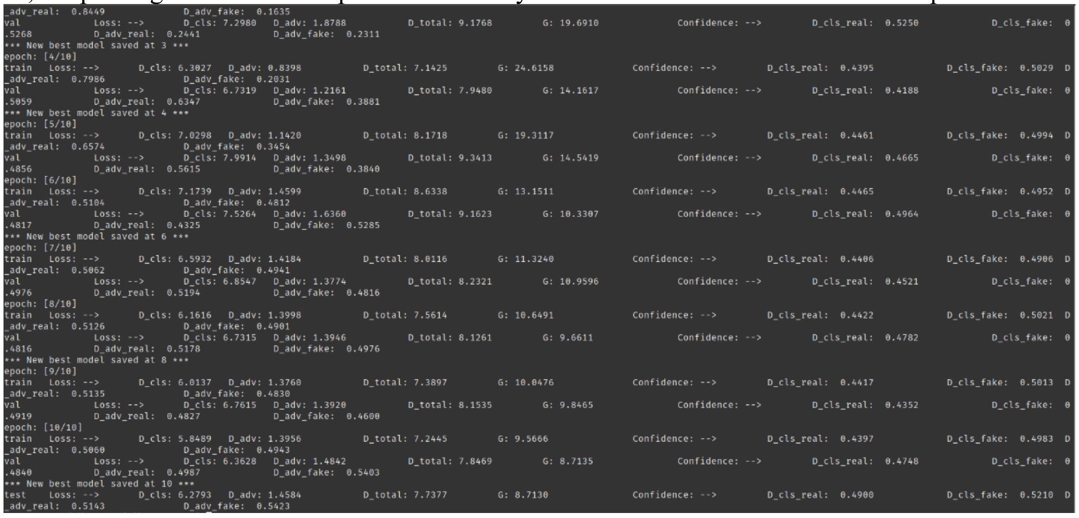
The redundancy of the model is measured by the cosine angle of the two active units. Finally, 35 hidden units are filtered out.

Although the speed of the pruned classifier on the test set is improved and the number of parameters is reduced, the accuracy of the model is reduced by about 0.1.

3.4 Result of Generative Modelling

According to the output, both accuracy and time needed are better compared with the pruning method. There are less testing loss and the shorter running time.

This may cause by that after rescaled and normalized, the figures lose some information and meanwhile decrease some noises. Also, the pruning method did not perform well may be due to the small number of nodes pruned.



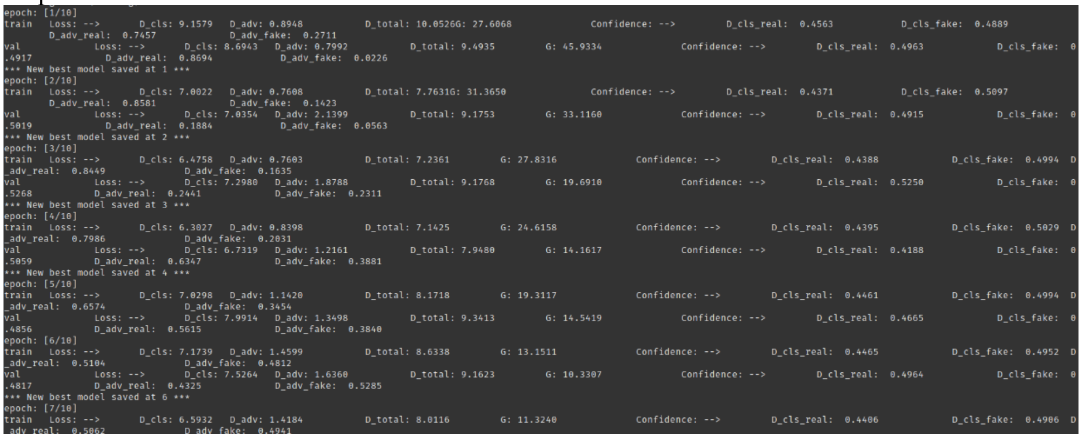
```
_adv_real:  0.8449              D_adv_fake:  0.1635
val             Loss: -->       D_cls: 7.2980   D_adv: 1.8788           D_total: 9.1768         G: 19.6910              Confidence: -->         D_cls_real:  0.5250             D_cls_fake:  0
.5268           D_adv_real:  0.2441             D_adv_fake:  0.2311
*** New best model saved at 3 ***
epoch: [4/10]
train   Loss: -->       D_cls: 6.3027   D_adv: 0.8398           D_total: 7.1425         G: 24.6158              Confidence: -->         D_cls_real:  0.4395             D_cls_fake:  0.5029  D
_adv_real:  0.7986              D_adv_fake:  0.2031
val             Loss: -->       D_cls: 6.7319   D_adv: 1.2161           D_total: 7.9480         G: 14.1617              Confidence: -->         D_cls_real:  0.4188             D_cls_fake:  0
.5059           D_adv_real:  0.6347             D_adv_fake:  0.3881
*** New best model saved at 4 ***
epoch: [5/10]
train   Loss: -->       D_cls: 7.0298   D_adv: 1.1420           D_total: 8.1718         G: 19.3117              Confidence: -->         D_cls_real:  0.4461             D_cls_fake:  0.4994  D
_adv_real:  0.6574              D_adv_fake:  0.3454
val             Loss: -->       D_cls: 7.9914   D_adv: 1.3498           D_total: 9.3413         G: 14.5419              Confidence: -->         D_cls_real:  0.4665             D_cls_fake:  0
.4856           D_adv_real:  0.5615             D_adv_fake:  0.3840
epoch: [6/10]
train   Loss: -->       D_cls: 7.1739   D_adv: 1.4599           D_total: 8.6338         G: 13.1511              Confidence: -->         D_cls_real:  0.4465             D_cls_fake:  0.4952  D
_adv_real:  0.5104              D_adv_fake:  0.4812
val             Loss: -->       D_cls: 7.5264   D_adv: 1.6360           D_total: 9.1623         G: 10.3307              Confidence: -->         D_cls_real:  0.4964             D_cls_fake:  0
.4817           D_adv_real:  0.4325             D_adv_fake:  0.5285
*** New best model saved at 6 ***
epoch: [7/10]
train   Loss: -->       D_cls: 6.5932   D_adv: 1.4184           D_total: 8.0116         G: 11.3240              Confidence: -->         D_cls_real:  0.4406             D_cls_fake:  0.4906  D
_adv_real:  0.5062              D_adv_fake:  0.4941
val             Loss: -->       D_cls: 6.8547   D_adv: 1.3774           D_total: 8.2321         G: 10.9596              Confidence: -->         D_cls_real:  0.4521             D_cls_fake:  0
.4976           D_adv_real:  0.5194             D_adv_fake:  0.4816
epoch: [8/10]
train   Loss: -->       D_cls: 6.1616   D_adv: 1.3998           D_total: 7.5614         G: 10.6491              Confidence: -->         D_cls_real:  0.4422             D_cls_fake:  0.5021  D
_adv_real:  0.5126              D_adv_fake:  0.4901
val             Loss: -->       D_cls: 6.7315   D_adv: 1.3946           D_total: 8.1261         G: 9.6611               Confidence: -->         D_cls_real:  0.4782             D_cls_fake:  0
.4816           D_adv_real:  0.5178             D_adv_fake:  0.4976
*** New best model saved at 8 ***
epoch: [9/10]
train   Loss: -->       D_cls: 6.0137   D_adv: 1.3760           D_total: 7.3897         G: 10.0476              Confidence: -->         D_cls_real:  0.4417             D_cls_fake:  0.5013  D
_adv_real:  0.5135              D_adv_fake:  0.4830
val             Loss: -->       D_cls: 6.7615   D_adv: 1.3920           D_total: 8.1535         G: 9.8465               Confidence: -->         D_cls_real:  0.4352             D_cls_fake:  0
.4919           D_adv_real:  0.4827             D_adv_fake:  0.4600
epoch: [10/10]
train   Loss: -->       D_cls: 5.8489   D_adv: 1.3956           D_total: 7.2445         G: 9.5666               Confidence: -->         D_cls_real:  0.4397             D_cls_fake:  0.4983  D
_adv_real:  0.5060              D_adv_fake:  0.4943
val             Loss: -->       D_cls: 6.3628   D_adv: 1.4842           D_total: 7.8469         G: 8.7135               Confidence: -->         D_cls_real:  0.4748             D_cls_fake:  0
.4840           D_adv_real:  0.4987             D_adv_fake:  0.5403
*** New best model saved at 10 ***
test    Loss: -->       D_cls: 6.2793   D_adv: 1.4584           D_total: 7.7377         G: 8.7130               Confidence: -->         D_cls_real:  0.4900             D_cls_fake:  0.5210  D
_adv_real:  0.5143              D_adv_fake:  0.5423
```

Figure 3. Result of generative modelling

3.5 Result of U-Net

An unexpected result is that adding U-net method didn’t show great improvement compared with generative modelling. According to the result, the accuracy is only improved by 0.17 and the time consumed is slightly improved.

This situation may be because I simply used resize for up-sampling, a transposed convolution model may perform better in this problem.



```
epoch: [1/10]
train   Loss: -->       D_cls: 9.1579   D_adv: 0.8948           D_total: 10.0526G: 27.6068              Confidence: -->         D_cls_real:  0.4563             D_cls_fake:  0.4889
        D_adv_real:  0.7457             D_adv_fake:  0.2711
val             Loss: -->       D_cls: 8.6943   D_adv: 0.7992           D_total: 9.4935         G: 45.9334              Confidence: -->         D_cls_real:  0.4963             D_cls_fake:  0
.4917             D_adv_real:  0.8694             D_adv_fake:  0.0226
*** New best model saved at 1 ***
epoch: [2/10]
train   Loss: -->       D_cls: 7.0022   D_adv: 0.7608           D_total: 7.7631G: 31.3650               Confidence: -->         D_cls_real:  0.4371             D_cls_fake:  0.5097
        D_adv_real:  0.8581             D_adv_fake:  0.1423
val             Loss: -->       D_cls: 7.0354   D_adv: 2.1399           D_total: 9.1753         G: 33.1160              Confidence: -->         D_cls_real:  0.4915             D_cls_fake:  0
.5019             D_adv_real:  0.1884             D_adv_fake:  0.0563
*** New best model saved at 2 ***
epoch: [3/10]
train   Loss: -->       D_cls: 6.4758   D_adv: 0.7603           D_total: 7.2361         G: 27.8316              Confidence: -->         D_cls_real:  0.4388             D_cls_fake:  0.4994  D
_adv_real:  0.8449              D_adv_fake:  0.1635
val             Loss: -->       D_cls: 7.2980   D_adv: 1.8788           D_total: 9.1768         G: 19.6910              Confidence: -->         D_cls_real:  0.5250             D_cls_fake:  0
.5268           D_adv_real:  0.2441             D_adv_fake:  0.2311
*** New best model saved at 3 ***
epoch: [4/10]
train   Loss: -->       D_cls: 6.3027   D_adv: 0.8398           D_total: 7.1425         G: 24.6158              Confidence: -->         D_cls_real:  0.4395             D_cls_fake:  0.5029  D
_adv_real:  0.7986              D_adv_fake:  0.2031
val             Loss: -->       D_cls: 6.7319   D_adv: 1.2161           D_total: 7.9480         G: 14.1617              Confidence: -->         D_cls_real:  0.4188             D_cls_fake:  0
.5059           D_adv_real:  0.6347             D_adv_fake:  0.3881
*** New best model saved at 4 ***
epoch: [5/10]
train   Loss: -->       D_cls: 7.0298   D_adv: 1.1420           D_total: 8.1718         G: 19.3117              Confidence: -->         D_cls_real:  0.4461             D_cls_fake:  0.4994  D
_adv_real:  0.6574              D_adv_fake:  0.3454
val             Loss: -->       D_cls: 7.9914   D_adv: 1.3498           D_total: 9.3413         G: 14.5419              Confidence: -->         D_cls_real:  0.4665             D_cls_fake:  0
.4856           D_adv_real:  0.5615             D_adv_fake:  0.3840
epoch: [6/10]
train   Loss: -->       D_cls: 7.1739   D_adv: 1.4599           D_total: 8.6338         G: 13.1511              Confidence: -->         D_cls_real:  0.4465             D_cls_fake:  0.4952  D
_adv_real:  0.5104              D_adv_fake:  0.4812
val             Loss: -->       D_cls: 7.5264   D_adv: 1.6360           D_total: 9.1623         G: 10.3307              Confidence: -->         D_cls_real:  0.4964             D_cls_fake:  0
.4817           D_adv_real:  0.4325             D_adv_fake:  0.5285
*** New best model saved at 6 ***
epoch: [7/10]
train   Loss: -->       D_cls: 6.5932   D_adv: 1.4184           D_total: 8.0116         G: 11.3240              Confidence: -->         D_cls_real:  0.4406             D_cls_fake:  0.4906  D
_adv_real:  0.5062              D_adv_fake:  0.4941
```

Figure 4. Result of U-net

3.6 Comparison with the Previous Works

According to previous paper, the weight significance of static weight matrix is not enough to be used as a tool to distinguish the hidden neuron function in this task, so the author is forced to continue to use more expensive methods, which use the measurement based on dynamic network behavior [2], and from my results the differences between two models are relatively small, as the accuracy is even reduced by 0.1.

Then according to another reference paper, the results of the pruned model are similar to the original model. An obvious improvement is that the computation time decreases gradually with the tailoring of the model [3]. This is also proved in my result as the pruned classifier speed is improved and the number of parameters is reduced and the accuracy is similar to the model without pruning.

The differences between my model and those used in previous papers may be that mine is relatively simpler, as I only used a regular three-layer model with simple optimization, and thus the number of redundant units may still not enough, Thus, in future works we can try to improve model and may get better outcome.

The generative modeling may perform better than my previous simple model, but it is still mostly following the result of reference paper, after adding the U-net method as the improvement of running time is more significant than that of accuracy.

4 Conclusion and Future Work

Through the pruning method, it can be seen that the pruned model can basically maintain the original performance, and on this basis, the number of parameters is reduced, the storage space is saved, and the calculation time is reduced. This technology can not only remove the redundant hidden layer neurons on the basis of the original model, but also provide a reference for the design of hidden layer neurons of a given data set in the process of neural network design. It should be noted that the processing of hidden layer neurons is limited to all neurons in the same hidden layer, not hidden layer cells in the same layer, and can not be used in the above methods. In the above process, the new model does not need retraining. In practical application, the same training set can be used to retrain the pruned model. Generally speaking, the fitting degree of the model to the data can be slightly improved.

The generative modeling has greater improved both running time and accuracy compared with simple pruning method. But after adding U-net, the improvement is slight compared with pure generative model, especially for the accuracy.

Through autoencoder and pruning technology, we find that both models and data have different degrees of redundancy. We use autoencoder to compress the data, and then prune to reduce the redundant activation units of the model. Unfortunately, the pruning reduces the accuracy of the model by nearly 0.1. In future work, we may further study this issue. Try to reduce the decrease in accuracy. And for the U-Net model, we may use transposed convolution to replace the resize function and see if there are better improvements.

References

1. Yao, Yue & Plested, Jo & Gedeon, Tom. (2018). Deep Feature Learning and Visualization for EEG Recording Using Autoencoders: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13–16, 2018, Proceedings, Part VII. 10.1007/978-3-030-04239-4_50.
2. Gedeon, T. D. . (1995). Indicators of hidden neuron functionality: the weight matrix versus neuron behaviour. 2nd New Zealand Two-Stream International Conference on Artificial Neural Networks and Expert Systems (ANNES '95), November 20-23, 1995, Dunedin, New Zealand. IEEE.
3. T. Gedeon, D. Harris. (1991). Network reduction techniques. Proceedings International Conference on Neural Networks Methodologies and Applications, 1991, vol. 1, pp. 119-126.
4. J. S. Rahman, T. Gedeon, S. Caldwell, R. Jones, M. Z. Hossain and X. Zhu. (2019). Melodious Micro-frissons: Detecting Music Genres From Skin Response. 2019 International Joint Conference on Neural Networks (IJCNN), 2019, pp. 1-8, doi: 10.1109/IJCNN.2019.8852318.
5. S. Lange and M. Riedmiller. (2010). Deep auto-encoder neural networks in reinforcement learning. The 2010 International Joint Conference on Neural Networks (IJCNN), 2010, pp. 1-8, doi: 10.1109/IJCNN.2010.5596468.
6. Kingma, D., & Ba, J. (2015). ADAM: A METHOD FORSTOCHASTICOPTIMIZATION [Ebook]. San Diego: The 3rd International Conference for Learning Representations.
7. YIN, X., GOUDRIAAN, J., LANTINGA, E., VOS, J., & SPIERTZ, H. (2003). A Flexible Sigmoid Function of Determinate Growth [Ebook] (pp. Pages 361–371). Annals of Botany, Volume 91, Issue 3.
8. Ronneberger, Olaf & Fischer, Philipp & Brox, Thomas. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. LNCS. 9351. 234-241. 10.1007/978-3-319-24574-4_28. YIN, X., GOUDRIAAN, J., LANTINGA, E., VOS, J., & SPIERTZ, H. (2003). A Flexible Sigmoid Function of Determinate Growth [Ebook] (pp. Pages 361–371). Annals of Botany, Volume 91, Issue 3.
9. Iglovikov, V., & Shvets, A. (2018). Ternausnet: U-net with vgg11 encoder pre-trained on imagenet for image segmentation. arXiv preprint arXiv:1801.05746.