Bidirectional Neural Network Transfer from eGC to Raw Dataset with Feature Extraction

Zishan Qin,
Research School of Computer Science,
Australian National University, Canberra, Australia
u6808226@anu.edu.au

Abstract. Extended Granger Causality (eGC) is a method for inferring causal relationships between pairs of nodes, which considers both lag and instantaneous effects. In this paper, I investigated to what extent the classification of eGC datasets using BiDirectional Neural Networks (BDNN) aids in the classification of the original thermal dataset in transfer learning. I also applied six common feature extraction methods to the original dataset to compress the original large features before performing classification and found that the one with linear discriminant analysis (LDA) works best on exploiting the transferred knowledge, gaining an average test accuracy of about 88.89%.

Keywords. BDNN, facial thermal imaging, Granger causality, Extended Granger causality, Knowledge transfer, transfer learning, Feature extraction, Principal component analysis (PCA), Independent component analysis (ICA), Linear Discriminant Analysis, Locally Linear Embedding, t-distributed stochastic neighbor embedding (t-SNE), Auto-encoder

1 Introduction

Bidirectional neural networks (BDNN) is a novel solution to the classification problem that can be trained without the constraints of input information, using only a predefined future framework (A. F. et al., 1995; Schuster et al., 1997; Collobert et al., 2004; Rakitianskaia et al., 2016), which is achieved by training in both positive and negative temporal directions.

However, if we use BDNN to classify a dataset with many features, since not every feature has a positive effect on the classification, we may be influenced by individual features and may wastefully train similar features, thus obtaining a poorly fitted model and subsequently a bad classification result. Therefore, we sometimes need to apply other deep learning methods to assist in solving the classification problem. In this paper, I applied transfer learning by transferring the parameters of the BDNN trained on the extended Granger Causality (eGC) dataset, expecting to get a better classification result for the original large dataset. My motivations are twofold:

First, it provides a new direction for the classification problem for large datasets. Since the original dataset contains so many features, some of which are not even useful, it is very complicated to classify it, usually taking a long computation time and requiring complex neural networks to fit. In contrast, the post-eGC dataset is highly condensed, reducing the 480 features of the original dataset to 20 features. By transferring the effective information and applying it, the amount of features required for the original large dataset can be greatly reduced, and the training can be accelerated to obtain a well-fitting model.

Second, it allows us to better understand the deeper meaning of statistical concepts on the original data. What exactly does a highly condensed statistical concept such as eGC entail about our original dataset? To what extent can this information help us understand the original dataset? This question can be answered by transfer learning of our classification problem.

I also apply six different feature extraction methods, including Principal Component Analysis (PCA), Independent Component Analysis (ICA), linear discriminant analysis (LDA), local linear embedding (LLE), t-distributed random neighborhood embedding (t-SNE) and Auto-Encoder (AE), before performing classification. An investigation was conducted on whether the feature extraction methods help with learning or hinder the learning. After comparison, the classification using transferred knowledge with LDA extraction shows up to have the best and the most stable results, with an average test accuracy of about 88.89%.

2 Dataset and Method

The main goal of this paper is to find an efficient way to transfer the latent information from the classification of the eGC dataset to the label prediction of the large thermal dataset. I describe these two datasets in section 2.1 and the methods that I used in section 2.2.

2.1 Description of the Datasets

The original dataset (Derakhshan et al., 2019) was collected in a hypothetical stressful crime experiment scenario in which participants were divided into two groups: a deception group, where those in the deception group would tell a lie, and an

honesty group, where those in the truthful group would tell a truth. A thermal imaging camera was simultaneously used to record the changes in facial temperature of each participant. By collecting temperature information from five facial regions, including periorbital, forehead, cheek, perinasal, and chin, the collected facial thermography footage was tracked. That's where the original dataset is from. It contains 480 features for 31 objects, because there're 31 participants and the camera's detector is with a maximum focal plane array (FPA) resolution of 480 dimensions.

For the eGC dataset, based on the raw dataset, the valid connections between these time series were calculated by an extended version of the GC method and provided as input to four linear and non-linear classifier models. This eGC is calculated using the following formula (1):

$$eGC_{Y\to X} = ln\frac{var(W_j(t))}{var(W_j'(t))}. \quad (1)$$

The effective connectivity between each paired node (ROI) were extracted by eGC, which created 20 features for 31 subjects with thermal videos and five ROIs for each face.

2.2 Method and Implementation

Generally speaking, I have applied BDNN, transfer learning and feature extraction in this research. In this section, the preprocessing of each of them is shown in 2.2.1. Using the preprocessed eGC dataset, the BDNN is used for the classification work, the details of which are described in 2.2.2. Then, feature extraction is performed to obtain the most important features from the preprocessed raw dataset, as described in 2.2.3. Finally, the transferred knowledge is used for classification, as shown in 2.2.4. The tuning details are provided in 2.2.5. Since it is a complex process, I have drawn a schematic diagram in Figure 1 for demonstration.

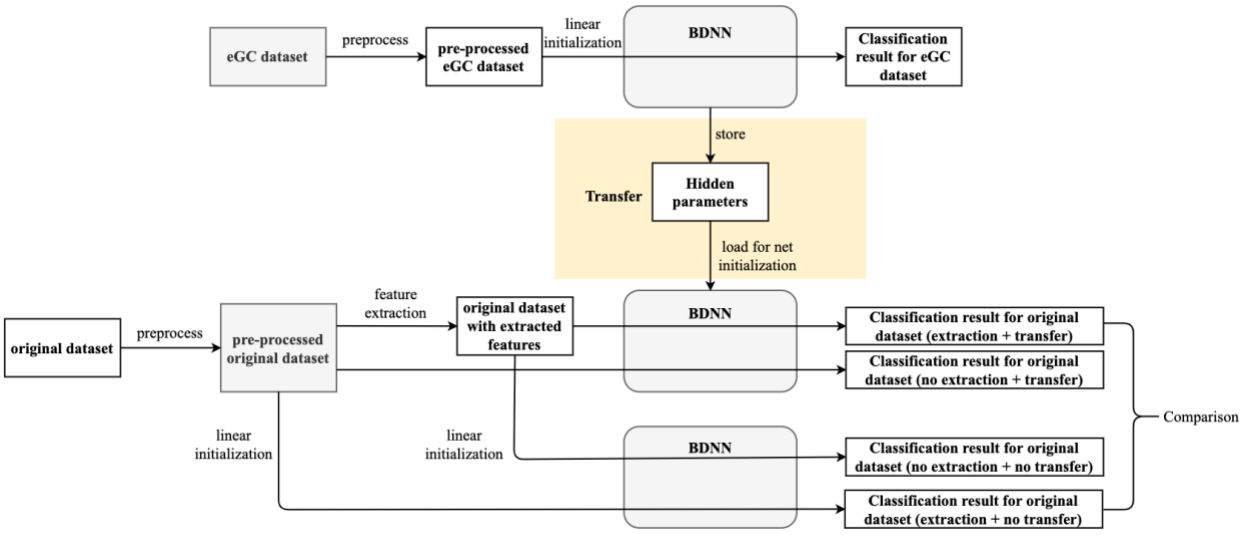

Figure 1. Structure of My Model

2.2.1 Preprocessing of the Datasets

The two datasets are a bit different. For the original dataset, the first column is from P1 to P31, specifying from whom the data is collected, while the eGC one does not have the column. Thus, I explicitly abandon this column for the original dataset. And then extracted the labels, calculated the average of each feature and save them back to the dataset as two single columns.

I have plotted some statistical values for each feature in the two datasets, including the mean, variance, minimum and maximum values. Specifically, I printed the maximum difference between means and variances for the two. For the eGC dataset, all values are in the range between 0 and 1, meaning that they have been normalized. While for the original dataset, the maximum difference between means is very large.

When using back propagation networks, it is necessary to normalize the data used for training according to the activation functions of the neurons for two main reasons: firstly, to make the input distribution of the different features vary little in order to prevent the classification of the overall features from being adversely affected by those features with relatively large

values; secondly, to adapt the variation interval of the output variables to the maximum output range of the corresponding activation function. According to (J. Sola & J. Sevilla, 1997), standardization can improve the performance of the model to between 5 and 10 folds, and the computational time required during training can be reduced by an order of magnitude.

```
# Check original dataset
check('original_dataset.xlsx')
```

```
Maximum difference between means:     0.9789
Maximum difference between variances: 0.0097
```

```
# Check eGC dataset
check('eGC_dataset.xlsx')
```

```
Maximum difference between means:     0.1164
Maximum difference between variances: 0.0611
```

Figure 2. Dataset Inspection

```
Before
Maximum difference between means:     0.9789
Maximum difference between variances: 0.0097

After
Maximum difference between means:     0.1165
Maximum difference between variances: 0.0292
```

Figure 3. Normalization Result for Original Dataset

Therefore, I did normalization using the normalize function from sklearn.preprocessing library for the original dataset. The result after normalization is shown in Figure 3. As can be seen from the figure, the data is much more balanced than before. And then, I split the data into train and test data by randomly splitting at a ratio of 70/30. In addition, I recorded the mean values of the features as bias values within the hidden layer of the training set for further processing to improve performance.

2.2.2 BDNN

The training model utilized here is BDNN (A. F. et al., 1995). A BiDirectional Neural Networks (BDNN) connects two hidden layers in opposite directions to the same output. There is one input at each moment, the hidden layer has two nodes, one for forward pass and the other for backward propagation, and the output layer is determined by the values of these two nodes, as shown in Figure 4.

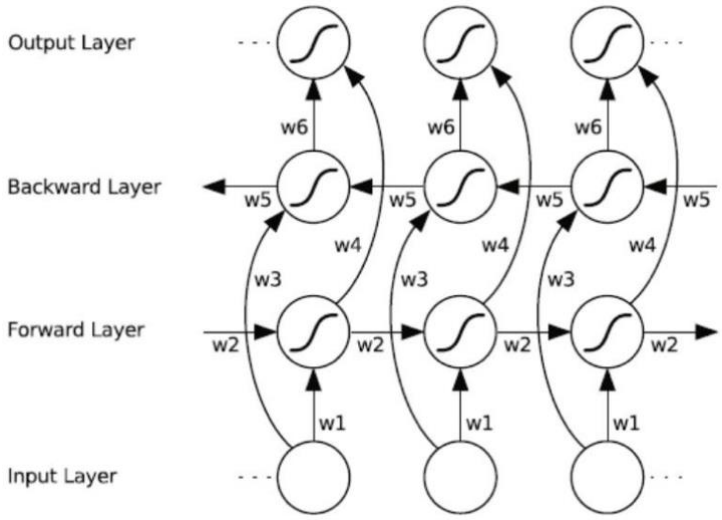

Figure 4. General Structure of BDNN

This model should be applicable because this structure provides the output layer with complete past and future information for each point in the input sequence (Schuster et al., 1997; Rakitianskaia et al., 2016). As in our dataset, each value corresponds to a relation between two subjects, then all value should have some relationship in between, which can be regarded as the past and future input information.

My implementation of BDNN can be summarized into the following pseudocodes:

Algorithm: BDNN

---

1) Forward Pass

for i = 1 to n **do**

Forward pass for the forward hidden layer.

Store the activations.

```
End for
for i = n to 1 do
      Forward pass for the backward hidden layer.
      Store the activations.
End for
for i = 1 to n do
      Forward pass for the output layer, using the stored activations from both hidden
layers.
End for
```

2) Backward Pass

```
for i = 1 to n do
      Backward pass for the output layer.
      Store the activations.
End for
for i = n to 1 do
      Backward pass for the forward hidden layer, using the stored activations from the
output layer
End for
for i = 1 to n do
      Backward pass for the backward hidden layer, using the stored activations from the
output layer
End for
```

3) Update Weights

Note that in my implementation, I use different loss functions for forward and backward pass. For forward pass, I use cross entropy loss function but for the backward pass, I use a loss function combining a Sigmoid layer and the BCE Loss in one single class, calculated by equation (2). The reason for doing this is to take the benefit of the log-sum-exp trick for numerical stability (Murphy, & K. P., 2006).

$$l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right] \quad \textbf{(2)}$$

2.2.3 Feature Extraction

In this paper, six feature extraction methods (Ippolito, 2019) are used, which are Principal Component Analysis (PCA), Independent Component Analysis (ICA), linear discriminant analysis (LDA), local linear embedding (LLE), t-distributed random neighborhood embedding (t-SNE) and Auto-Encoder (AE).

Principal Component Analysis. PCA is one of the most common and traditional linear dimensionality reduction techniques. In PCA, the original data is the input, and the goal is to try to find a combination of input features that best summarizes the distribution of the original data and thus reduces its original dimensionality. PCA can maximize the variance and minimize the reconstruction error by looking at the distance of the pairs. In PCA, our raw data is projected onto a set of orthogonal axes, with each axis ranked by importance. The reason this approach may work is that out of 480 features in the original dataset, we were able to select the top 20 most important features as representatives for subsequent operations. The reason I chose the number of 20 here is that there are also 20 features in the eGC dataset, and I wanted to make the dataset before and after the transfer relatively similar.

Independent Component Analysis. ICA [10] is a linear dimensionality reduction method that takes a mixture of independent components as input data, with the aim of correctly identifying each component and removing all unnecessary noise. Two input features are considered independent if their linear and nonlinear dependencies are both equal to zero. The reason why this approach may be applicable is that it removes those noises from the 480 features of the original dataset that affect the correct classification and may be useful for our subsequent processing. In the same way as the PCA selection, in our experiments, after ICA we will also select only 20 features.

Linear Discriminant Analysis. Unlike the first two methods, LDA is a supervised learning dimensionality reduction technique and machine learning classifier that aims to maximize the distance between the means of each class and minimize the spread of the classes themselves. Therefore, LDA uses both within-class and between-classes as measures. This is a good choice because maximizing the distance between the means of each class can lead to better classification results when

projecting the data into a low-dimensional space, due to the reduced overlap between different classes. LDA can have good results when the input data follows the Gaussian distribution, which is applicable since our original dataset has been normalized. Here, the feature dimension after LDA is reduced to 1.

Local Linear Embedding. Methods such as PCA and LDA can perform very well when there is a linear relationship between different features, but the original dataset is not necessarily linear between features, so we also need to consider how to handle the nonlinear case. LLE is a dimensionality reduction technique based on Manifold Learning, aiming to make this object representable in its original D-dimensional space, rather than in an unnecessarily larger space. It seeks a low-dimensional projection of the data, preserving distances within local neighborhoods. It can be considered as a sequence of local PCA to find the best nonlinear embedding by global comparison. Here the number of features after LLE is reduced to 20.

t-distributed random neighborhood embedding. t-SNE is a nonlinear dimensionality reduction technique that works by minimizing the difference between a distribution consisting of pairwise probabilistic similarities of input features in the original high-dimensional space and the equivalent distribution in the reduced low-dimensional space, using gradient descent to minimize the KL divergence. The dimension after t-SNE is 3.

Auto-Encoder. AE is composed of an encoder and decoder. This dimensionality reduction method involves learning a representation (encoding) for a set of data, by training the network to ignore signal "noise". While performing the dimensionality reduction and learning the reconstruction side, the autoencoder tries to produce a representation from the reduced encoding that is as close as possible to the original input. It can ignore a portion of the noise of the features, which is why I utilize this method for dimensionality reduction here. The result of the dimensionality reduction is in three dimensions.

In section 3.2, I will analyze the result for each and explore which method aids the exploitation of transferred knowledge the most.

2.2.4 Transfer Learning

In deep learning, transfer learning (Weiss et al., 2016) is a technique in which a neural network model is first trained on a problem, and then the hidden layers from the trained model are used as initialization for a new model trained on a new problem. The information from these hidden layers can be effectively used in the new problem, and a well-fitting model can be obtained early in the training. We call the previous problem the source domain and the new problem later the target domain. Then, in transfer learning, we want to explore an effective transfer method so that the knowledge transferred from the source domain can help the training in the target domain.

In my research, the source domain is the problem of classifying the eGC dataset, the target domain is the problem of classifying the original dataset (which may go through the feature extraction step), and the transferred knowledge is the hidden layer of the BDNN trained on the eGC dataset.

2.2.5 Hyper-Parameter Tuning

The tuning is on the original dataset after extracting features by LDA. There're five hyperparameters for adjustment: number of classes, learning rate, hidden layer nodes' number, number of epochs and batch size. For this experiment, the number of classes is fixed, and is equals to 2. For the rest of the hypermeters, I have designed some simple experiments for tuning.

For tuning the learning rate, one strategy is to start with a larger value and reduce it to near zero at the end of training. Therefore, based on the dataset not being very large, I designed an experiment that started at 0.5 and reduced it by 0.01 each time until it touched zero, comparing the results of three parallel experiments for the average test accuracy. The results show that the best learning rates for the three experiments were 0.39, 0.43 and 0.42 (Table 1). Therefore, I set the learning rate to be the mean, i.e., 0.41.

Table 1. Tuning learning rate experiment result (hidden size = 10, number of epochs = 100, batch size = 16)

	Highest test accuracy	Best learning rate
Exp. 1	88.89%	0.39
Exp. 2	88.89%	0.43
Exp. 3	88.89%	0.42
Average	88.89%	0.41

To find the most suitable number of nodes for the hidden layer, as it should lie between the input size and the output size, I also tested using three parallel experiments, starting with the size of the training set and decreasing it by one at a time until it reached 0. The experiment was repeated three times and the size with the maximum test accuracy was recorded. A summary of the test result is shown in Table 2. Therefore, I choose the hidden size to be 5.

Table 2. Tuning hidden layer node number experiment result (learning rate = 0.41, number of epochs = 100, batch size = 16)

	Highest test accuracy	Best hidden size
Exp. 1	88.89%	2
Exp. 2	88.89%	3
Exp. 3	88.89%	11
Average	88.89%	5

Similar experiments were conducted on batch sizes, with a summary of the results shown in Table 3. there are typically three choices of batch size: batch mode, where the batch size is equal to the total dataset so that iterations and epoch values are equal; mini-batch mode, where the batch size is greater than 1 but less than the total dataset size; and stochastic mode, where the batch size is equal to 1, and the gradient and neural network parameters are updated after each sampling [9]. I therefore started with a batch size of 1 until it was equal to the size of the dataset, in order to find the parameter that would give the maximum test accuracy. From result shown in Table 3, I choose the batch size to be 14.

Table 3. Tuning batch number experiment result (learning rate = 0.41, hidden layer size = 5, number of epochs = 100)

	Highest test accuracy	Best batch size
Exp. 1	88.89%	9
Exp. 2	88.89%	7
Exp. 3	88.89%	10
Average	88.89%	9

For the epoch size adjustment, I first initialized it to the dimensionality of the training set and then increased it by 10 at a time until it converged to a value where the difference between the current accuracy and the previous accuracy was less than 1e-12, until it reaches 500 epochs. I summarize the results in Table 4 and we could find that the number of epochs is set to 100 is appropriate, so I choose the number of epochs to be 100.

Table 4. Tuning epoch number experiment result (learning rate = 0.41, hidden layer size = 5, batch size = 9)

	Best epoch number
Exp. 1	130
Exp. 2	90
Exp. 3	80
Average	100

Same methods are performed on eGC dataset and the summary of the tuning results is shown in Table 5. As the hidden layer size should be the same, so I let the eGC dataset's hidden_size also be 5.

Table 5. Hyperparameter result

Original with LDA	Num_classes	Learning_rate	Hidden_size	Num_epochs	Batch_size
	2	0.41	5	100	9
eGC dataset	Num_classes	Learning_rate	Hidden_size	Num_epochs	Batch_size
	2	0.085	~~21~~ **5**	100	14

3 Results and Discussion

Based on the above, I designed seven groups of experiments, six of which are experimental groups applying different feature processing methods to the original dataset, and the control group is a classification without applying feature processing methods, as detailed in section 3.1. After having found the feature extraction method that best fits the model, I performed a statistical analysis of my results, and the statistical indicators are described in section 3.2. Finally, all my experimental results are summarized, and the issues included in my study are described in section 3.3.

3.1 Effect of Different Feature Extraction Methods

Based on the purpose of finding the best method to classify the original large dataset by transferring the neural network parameters of the eGC dataset trained with BDNN, I investigate the impact of the six different feature methods described in the previous section on the transfer by designing 100 sets of parallel experiments and counting two metrics, the average test accuracy and the cumulative test accuracy for the first 50 sets of each group. The reason I use cumulative test accuracy here is that training can be unstable, and by being cumulative can give a good indication of the general trend of the neural network

results. Note that these experiments were performed with hyperparameters under the eGC dataset rather than after retuning, so the test accuracies are not very high. Nevertheless, it is sufficient to show the improvement when using the transfer network.

For the control group, the results are shown in Figure 5. It is clear that transfer learning helps to obtain a better model with higher test accuracy than training from scratch. For the other experimental groups, the results are shown in Figure 6 by showing the average test accuracy, and in Figure 7 by showing the cumulative test accuracy for 50 sets of parallel experiments.

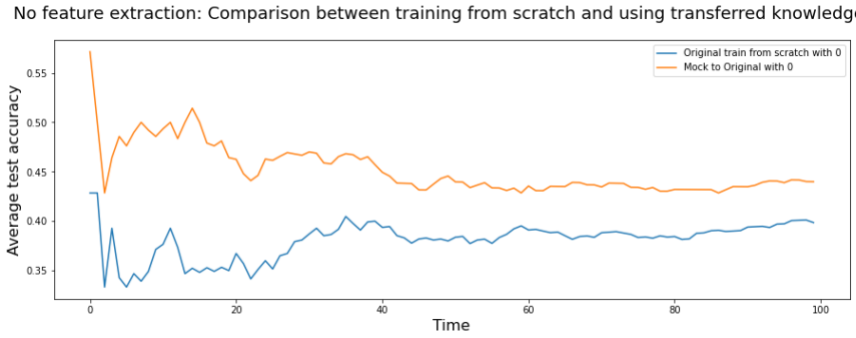

Figure 5. Training from Scratch vs Using Transferred Latent Space

As can be seen from the Figure 6, PCA, LDA and TSNE could gain positive transfer effects, while the rest gained negative transfer results. From the three methods with positive results, LDA gained the most significant and stable improvement, from Figure 7, because it is the only method with obvious orange and blue lines above the benchmarks'. Therefore, we can know that supervised feature extraction could get better results compared to the unsupervised ones'.

Figure 6. Average Test Accuracy for the Six Feature Extraction Methods

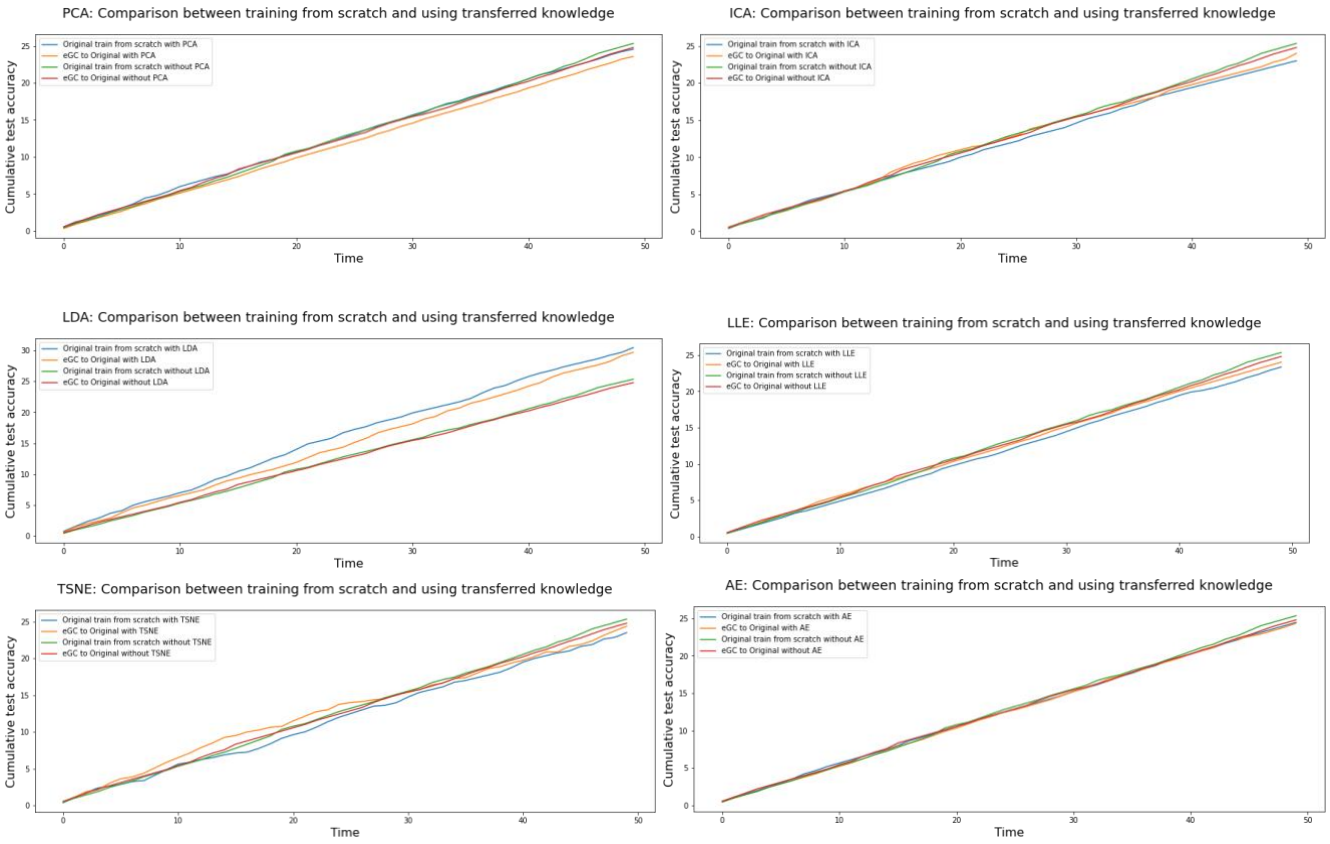

Figure 7. Cumulative Test Accuracy for the Six Feature Extraction Methods

3.2 Indicators for statistical comparison

There're in total 6 indicators using for comparisons: test loss, test accuracy, precision, recall, specificity, and F-score. Loss value can be seen as the difference between the true values and the predicted values by the model, which could imply how poorly or well a model behaves after each iteration of optimization. An accuracy metric is used to measure the percentage that the model makes a correct prediction, and it tells about the performance of the algorithm. In classification, both precision and recall are about relevance. Precision is the fraction of relevant instances among the retrieved instances while recall (also known as sensitivity) evaluates the relevant instances that were retrieved. Specificity is a measure of how well a test can identify true negatives. And F-score, the harmonic mean of precision and recall, is another measure of a test's accuracy, calculated by equation (3):

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \qquad \textbf{(3)}$$

3.3 Overall Performance, Comparisons and Analysis

After adjusting the hyperparameters, I use LDA for feature extraction and then use the transferred knowledge for classification to get the final result. The other three benchmarks are: with feature extraction but not using transferred knowledge for classification, without feature extraction and using transferred knowledge, and without feature extraction and not using transferred knowledge.

By comparison in Table 6, we can see that using LDA and transfer gets the best result in both test accuracy and test loss, much better than the statistics provided in [5], followed by using LDA and train from scratch, which gets the same test accuracy but with a larger loss. This shows that the feature extraction did aid the transfer learning for classification. I think the reason is that, after normalization, the original dataset basically follows the Gaussian distribution, which gives LDA good result for choosing valuable features from the large amount of data.

For the two without LDA, the transfer one gets a worse result. This is the first limitation of my research, and I think the reason may be that, without feature extraction and directly transfer would make the network overfitting with the training of the eGC

dataset and therefore could not fit with the original dataset well. Another issue with my research is that my method with LDA for feature extraction will take longer for training, compared with the benchmark. Although less features existed in the dataset, but more time will be needed for fitting. Last but not least, my result is still unstable enough. This may be because the transfer model is not fitted well enough.

Table 6. Results comparisons (the first chart is to compare with the original paper in [5]; the second one is to compare with my benchbacks)

	My result	Baseline 1 in [5]: Without Feature Extraction	Baseline 2 in [5]: KNN	Baseline 3 in [5]: LDA	Baseline 4 in [5]: SVM Linear
Accuracy (%)	88.89	67.10	82.05	84.6	87.1
Sensitivity	0.9	Not given	0.9	0.9	0.9
Specificity	0.5	Not given	0.736	0.789	0.842
Precision	0.9	Not given	0.782	0.818	0.857
F-Score	0.8889	Not given	0.837	0.857	0.878

	LDA + Transfer	LDA + Train from scratch	Without LDA + Transfer	Without LDA + Train from scratch
Accuracy (%)	88.89	88.89	55.56	55.56
Loss	0.6207	0.6349	0.8930	0.6933
Sensitivity	0.9	0.9	0.5556	0.5556
Specificity	0.5	0.5	0.5	0.5
Precision	0.9	0.9	0.5	0.5
F-Score	0.8889	0.8889	0.3572	0.3572
Time spent (seconds)		1.7973		1.6086

4 Conclusion and Future Work

From my experiments and implementation, I found that extracting features by using PCA, LDA and TSNE can help with positive transfer, showing that the three methods aid in using the latent information from the BDNN model from the eGC dataset. The best performance was achieved using LDA, with an average test accuracy of about 88.89%. However, if we do not use feature extraction, transfer learning may cause the model to overfit with the eGC dataset, such that the test result would be worse than training from scratch.

However, much work ahead remains to be done. One important issue is the instability of my model results, which may be due to the fact that my model does not fit well to the original thermal dataset. This can be further improved by applying different loss functions and using different vectors as hidden biases, etc. The second issue is that it will take more time than before. This can be further improved by simplifying the model of BDNN. Last but not least, it has a high risk of overfitting after the transfer. This can be addressed by applying dropout (Srivastava et al., 2014), "early stopping", "network reduction", "data expansion" and "regularization" (Ying, 2019).

References

1. Rakitianskaia, A., Bekker, E., Malan, K. M., & Engelbrecht, A.: Analysis of error landscapes in multi-layered neural networks for classification. 2016 IEEE Congress on Evolutionary Computation (CEC). IEEE, pp. 5270-5277 (2016)
2. A. F. Nejad, & T. D. Gedeon: Bidirectional neural networks and class prototypes. Proceedings of ICNN'95 - International Conference on Neural Networks, vol.3, pp. 1322-1327 (1995)
3. M. Schuster, & K. K. Paliwal: Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673-2681 (1997)
4. Collobert, R., & Bengio, S.: Links between perceptrons, MLPs and SVMs. Proceedings of the twenty-first international conference on Machine learning, p. 23 (2004, July).
5. Derakhshan, A., Mikaeili, M., Nasrabadi, A. M., & Gedeon, T.: Network physiology of ‘fight or flight’ response in facial superficial blood vessels. Physiological measurement. 40(1), 014002 (2019)

6. J. Sola & J. Sevilla: Importance of input data normalization for the application of neural networks to complex industrial problems. IEEE Transactions on Nuclear Science, vol. 44, no. 3, pp. 1464-1468 (June 1997)
7. BCEWithLogitsLoss — PyTorch 1.8.1 documentation. (2021)
8. Murphy, & K. P.: Naive bayes classifiers. University of British Columbia, 18(60) (2006)
9. Ippolito, P.: Feature Extraction Techniques. Towards data science. (2019) Accessed at https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be.
10. Diving Deeper into Dimension Reduction with Independent Components Analysis (ICA), Paperspace. Accessed at: https://blog.paperspace.com/dimension-reduction-with-independent-components-analysis/
11. Iterative Non-linear Dimensionality Reduction with Manifold Sculpting, ResearchGate. Accessed at: https://www.researchgate.net/publication/220270207_Iterative_Non-linear_Dimensionality_Reduction_with_Manifold_Sculpting.
12. Manifold learning, Scikit-learn documentation. Accessed at: https://scikit-learn.org/stable/modules/manifold.html#targetText=Manifold%20learning%20is%20an%20approach,sets%20is%20only%20artificially%20high.
13. Steven Flores.: Variational Autoencoders are Beautiful, Comp Three Inc. Accessed at: http://www.compthree.com/blog/autoencoder/.
14. Weiss, K., Khoshgoftaar, T. M., & Wang, D.: A survey of transfer learning. Journal of Big data, 3(1), 1-40 (2016)
15. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958 (2014)
16. Ying, X.: An overview of overfitting and its solutions. In Journal of Physics: Conference Series (Vol. 1168, No. 2, p. 022022). IOP Publishing. (2019, February)