Constractive LSTM-FCN for Few shot Time Series Classification

ZiMing Xu* GuoQi Yu* MuXuan Li *
University of Electronic Science and Technology of China
{2020080602015, 2020080602021, 2020080602022}@uestc.edu.cn

Abstract

In the paper, we use data scaling to preprocess the data and analyze the Squeeze-and-Excitation block we use in the network LSTMFCN about which channel it measures can have better effects. During the training, we design the clustering loss and the contrastive loss with the characteristics of plug and play to aid training and reduce overfitting. During the process, we can get the pseudo labels in the test set, we use the labels to the method called PWDBA(Pseudo Weighted DTW Barycentric Averaging), which can improve the stability of training.

***Keywords**: data preprocessing, Squeeze-and-Excitation, contrastive learning, Pseudo Weighted DTW Barycentric Averaging*

1. Introduction

n the fields of computer vision and natural language processing, the most mainstream data preprocessing methods are BatchNorm [1] and LayerNorm [2], respectively, which select a single neuron within a batch, and all neurons in a particular layer as the normalization direction to scale and process the data, respectively.

These data processing schemes effectively alleviate the problems of gradient explosion, gradient disappearance, etc. in neural networks and also make the network training more stable. In the field of deep learning, implicitly transforming the intermediate layer inputs in the network layers using some methods (e.g., normalization) is a standard and common approach to bringing them to the same numerical scale. Meanwhile, in univariate time series (one channel) problems, max-min, standard normalization, or zero-mean normalization of the data are more common processing schemes. Although these schemes are different in terms of processing means, the essence is to scale the data to better fit the training process.

In recent years, the analysis and processing of time series have gradually shifted from univariate (one-channel) to multivariate (multi-channel). In terms of model selection and use, several algorithms choose to apply a model to a multivariate univariate time series problem and then aggregate classification decisions using some integration or voting methods [3]. Similar to this approach, quite a few multivariate time series problems follow the same pattern of preprocessing the data inputs. For example, when classifying univariate time series data, we assume that the dimensionality of the data set is [samples, timesteps] (channels default to 1, so they are omitted), and thus data preprocessing and data augmentation for him is quite easy. Thus, it seems natural to apply the same transformation to each channel of a multivariate dataset with different channel dimensions [samples, channels, timesteps] (e.g., OCD, FTD, ADNI datasets in this experiment), treating each channel as a separate univariate sequence.

Also, in recent years, experiments have proved that FCNs have a high performance in TimeSeries Classification Tasks. The methods of constructing FCNs have correlation-based approaches, Granger causality analysis, regularized inverse covariance estimation, etc [4]. So we combine these methods and achieved a good result.

In the paper, we introduce the data preprocessing way of scaling to smooth the data preliminary. Then we use the LSTMFCN [5] as the base network architecture to extract the time feature and the special feature, however, we use Squeeze-and-Excitation Networks [6] to reduce the noise in the data as well as exploring the influence of using a different channel to make the SE operations. Due to some elements like noise, the classifier can not make the precise classification, we design the clustering loss to make the data have more obvious special features and get the pseudo labels. And we design the contrastive loss with the characteristics of plug and play using the pseudo label that can constrain the clustering and it is helpful to find better classification and reduce overfitting. After some epochs, the accuracy of pseudo labels is good enough and we use it for PWDBA(Pseudo Weighted DTW Barycentric Averaging) it can help to measure the datasets which can make training stabler. The remainder of the paper is organized as follows. Section II reviews the related work. Section III presents the architecture of the proposed methods. Section IV analyzes and discusses the

*Equal contribution

experiments performed. Finally, conclusions are drawn in Section V.

2. Related Work

2.1. Time Series Classification

Distance-based methods along with k-nearest neighbors have proven to be successful in classifying multivariate time series [7]. Plenty of research indicates Dynamic Time Warping (DTW) as the best distance-based measure. In addition to distance-based metrics, feature-based classification algorithms rely heavily on the features being extracted from the time series data [8]. However, feature extraction is arduous because intrinsic features of time series data are challenging to capture. Also, there may have lots of noise making it hard to obtain the characteristic feature. In a recent search, distance-based approaches are more successful in classifying multivariate time series data [9]. Nowadays, deep neural networks have been employed for time series classification tasks. Multi-scale convolutional neural network (MCNN) [10], fully convolutional network (FCN) [11], and residual network (ResNet) [11] are deep learning approaches that take advantage of convolutional neural networks (CNN) to extract feature and make classification. FCN and ResNet do not require any heavy preprocessing of the data or feature engineering. In [5], it improves the performance of FCN by augmenting the FCN module with either a Long Short Term Recurrent Neural Network (LSTM RNN) sub-module, called LSTM-FCN, or an LSTM RNN with attention, called ALSTM-FCN to visualize the Class Activation Maps (CAM) of the convolutional layers to detect regions that contribute to the class label. In [6], it utilizes the Long Short Term Memory Fully Convolutional Network (LSTM-FCN) and Attention LSTM-FCN (ALSTM-FCN), into a multivariate time series classification model by augmenting the fully convolutional block with a squeeze-and-excitation block to further improve accuracy.

2.2. Contrastive learning

Nowadays contrastive learning is widely used in different fields of deep learning. DeepInfomax formalizes this problem in a view of mutual information. MoCo [12] utilizes a memory bank and some implementation tricks to achieve good performance. SimClr [13] improves contrastive learning by using a larger batch size and data augmentation. Currently, [14] suggests that info N CE has better performance than cross-entropy on supervised classification. InfoPatch [15] repurposes the contrastive learning for such matching to learn a few-shot embedding model. It investigates contrastive learning with Noise Contrastive Estimation (NCE) in a supervised manner for training a few-shot embedding model and shows it is an effective way.

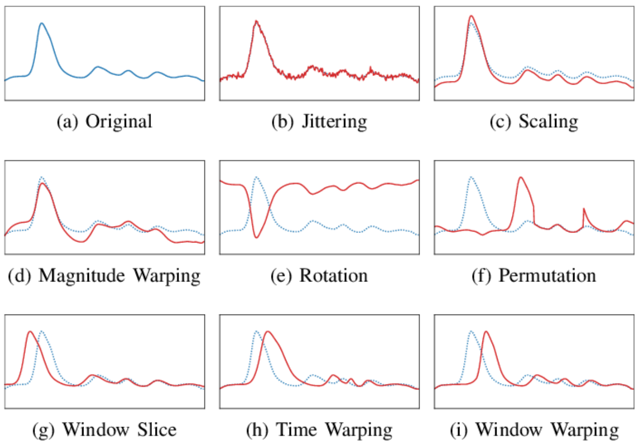


Figure 1. Some examples of time series data augmentation methods

2.3. Data Augmentation

Many researches show that the methods used for augmentation of image data in computer vision, such as Masking, Flipping, Rotating, Cropping, etc., can achieve good data enhancement performance and model accuracy improvement, however, as mentioned in [16], the dynamics of multivariate time series with time steps (timesteps) dynamics and spatial dependence between different channels and temporal dependence between different timesteps can pose completely different challenges. In [17], Brian Kenji Iwana et al. studied the effect of the following time series preprocessing means. They include, Jittering [18], Rotation [19], Scaling [18], Magnitude Warping [18], Permutation [18], Slicing [20], The path of Time Warping [18], Window Warping [20], SubоPtimAl Warped time series geNEratoR (SPAWNER) [21], Weighted DTW Barycentric Averaging (wDBA) [22], Random Guided Warping (RGW [23], Discriminative Guided Warping (DGW) [23].Each of these methods differs, and all serve to enhance the time series data in specific situations. The visualizations are shown in 1

Synthetic example augmentation has also been used to deal with unbalanced classes (i.e., when the classes in the training set do not have the same number of elements.) Chawla et al [19] proposed a method called SMOTE (Synthetic Minority Oversampling Technique), which creates synthetic minority class instances to balance the classes.

3. Method

Network architecture

Fully convolutional Networks are typically used as feature extractors to extract spatial features. Also, the LSTM block with the dropout can obtain the time feature. So we use the LSTMFCN [24] as a baseline by combining the benefits of these two architectures.

In the LSTMFCN, we use the FCN and the LSTM block to extract features of datasets and finally concatenate the feature then pass it into a softmax classification layer. And during the experiment, we found there is a lot of noise in datasets as well as features and the noise can make the network focus on the distracting features and cause overfitting. So we pay attention to how to reduce noise in the features so the model can increase the weight of the important features. We use ideas in Squeeze-and-Excitation Networks [6]. Also, the datasets are different from the datasets in the [6], like OCD_fMRI, each data have 90 brain areas and 200 time steps and both of them need to pay attention to. So we explore which channel has a heavier weight on the classification. In our LSTMFCN model, asides from the traditional cross-entropy loss, we also add a clustering loss to find a better feature space for the datasets. Then we utilize a contrastive loss to constraint the classification of the test datasets which can effectively prevent overfitting. WDBA [22] is a data preprocessing strategy which uses DTW to average the data feature and it is helpful to classify time series data.

In the rest of the section, we explain the way of data preprocessing in Sect 3.1. Then we talk about the network architecture in Sect 3.2. In Sect 3.3 we introduce the clustering loss and the contrastive loss. In Sect3.4, we introduce the PWDBA(Pseudo weight DTW Barycentric Averaging).

3.1. Data Scaling

Data scaling/preprocessing methods.

- **Normalization:** for each dimension of the vector X the data $X_1, X_2, ..., X_n$ are divided by $||X||2$ to obtain a new vector.
- **Standardization:** Z-Score normalized to a mean of 0 and a standard deviation of 1 (variance is also 1)
- **MinMax:** max-min normalization, scale the data to $[0, 1]$ space.
- **MaxAbs:** The data will be scaled to between $[-1, 1]$, which is the feature by which all data will be divided by the maximum value.
- **Robust:** scales the data according to quartiles. For cases where the data have a high number of outliers, it is clearly inappropriate to use the mean and variance to normalize the data, so robust scales the data by the median, first and quartiles, which is much better.
- **Power Transformation:** PowerTransformer provides nonlinear transformations where data is mapped to a normal Gaussian distribution to stabilize variance and minimize skewness.
- **Quantile Transformation:** provides nonlinear transformation where the distance between edge outliers and outliers is reduced and all data will be mapped in the range [0, 1]. It transforms the features to follow a uniform or normal distribution. Thus, for a given feature, this transformation tends to the most frequently scattered value.

Data Augmentation methods.

We first define a time series $T =< t_1, ..., t_{Li} >$, and a time series dataset $D = T_1, ..., T_N$, where L represents the length of the time series and Drepresents the data set consisting of a series of time series samples. The intuition behind our approach is to sample a small sample set from the original dataset, use it to generate a new time series data $\bar{T}$using the method of DTW, record it and put it back into the original data D. This way we get a larger dataset consisting of D merged on the just generated $\bar{T}$.Repeating the above process, we can obtain a series of $\bar{T}i$ using K-MEANS clustering algorithm, we denoted it as $\bar{D}$ and then replace the original dataset with this new, more robust sample set for training and testing. To control the parameters and facilitate training, we specify that the generated new sample set is equal to the size of the original dataset, i.e., $\bar{D} = D$. And for the generation of weights, we chose the Average Selected with Distance (ASD) criterion from the original paper [22].i.e. we first choose randomly a time series T^* from D and give it a weight of 1. We then assign its nearest neighbor a weight of 0.5. Next, we define an exponential decay function that maps a distance (DTW) to a weight. The more detailed function is below:

$$\omega_i = e^{\ln(0.5) \times \frac{DTW(T_i),T^*}{d^*_{NN}}} \tag{1}$$

where d^*_{NN} is the distance between T^* and its nearest neighbor. We then use this function and normalize its outputs to decide upon the weights for all time series in D. Intuitively, we make $T^*$the center of gravity of our synthetization and create a power law around it whose width is decided by how close the nearest neighbor is. If it is very close, then only elements that are extremely close to T^* and its nearest neighbor will influence the generation. If it is far, then many other time series will. Here, we initialize DBA with T^*.

3.2. LSTM-FCN architecture

The model we used is shown in fig.2 The fully convolutional block consists of three stacked temporal convolutional blocks with filter sizes of 128, 256, and 128 respectively. Each convolutional block consists of a temporal convolutional layer, which is accompanied by batch normalization followed by a ReLU activation function. Finally, global average pooling is applied following the final convolution block.

Also, the time series input is conveyed into a dimension shuffle layer. And then it is passed into the LSTM block followed by a dropout. Finally, The output of the global pooling

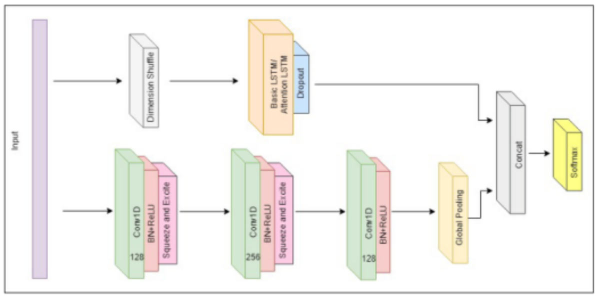


Figure 2. The architecture of LSTM-FCN.

layer and the LSTM block is concatenated to combine the spatial features and time features and passed onto a softmax classification layer. We find that there is a lot of noise in the data and so the feature extractor has some distracting features which may cause overfitting. So at the end of the first and the second FCN block, we add a squeeze-and-excite block. In the [25], it also uses the block to adaptively recalibrate the input feature maps. However, we have different settings because data has 90 brain areas and 200 time steps. So we explore which channel we should focus on. We find that using squeeze-and-excite block on time steps can have smaller variance in the same-label sample than on brain areas which means that it can better reduce the noise on time step.

At the same time, we try the different channels and the best outcome appears when we have this operation on the 200 time steps.

3.3. Cluster Loss and Contrastive Loss

During we use the original LSTMFCN to train with just CrossEntropy loss, we find that it easily has the problem of overfitting. We want to use a way to let the data have the same label cluster and leave far away from the data with a different label. If we can let the clusterings leave far enough, we can effectively reduce overfitting. So we design the clustering loss L_c and used it to train with the train set. It can help the train set to cluster. We use the L2 distance to find k closest sample features. And then we use clustering to make the same label sample get closer and make samples of different labels to far away. The loss is defined as:

$$\mathcal{L}_c = \sum_{i \in B} l_l^i \tag{2}$$

Given M_k and N_K are the set of k-neighbors of sample i, z_i is the feature of sample i which extract using the network, then $\mathcal{L}_l^i$ is defined as:

$$\mathcal{L}_l^i = -\log \frac{\sum\limits_{j \in M_k(i)} \exp \frac{z_i z_j}{||z_i|| ||z_j||}}{\sum\limits_{m=1}^{k} \exp(\frac{z_i z_m}{||z_i|| ||z_j||})} \tag{3}$$

Besides, some test set data is also in the boundary so it may not classify precisely. Also, after some epochs, the test set data will leave the boundary and the train set will be far away. So we want to use the test set feature to constraint and help the test set to cluster. We design the contrastive loss $\mathcal{L}_t$. So we use the neighbor samples label to obtain $\mathcal{L}_n$, and the output of the classifier to get L_c. We just use the sample whose $\mathcal{L}_n$ is identical to $\mathcal{L}_c$ to participate in the train and it can constrain the pseudo labels. We also use the L2distance to find k closest sample features. We consider the samelabel samples as positive samples and the differentlabel samples as negative samples. And then we utilize the contrastive loss to the cluster. And using this loss, the data in the test set can be left far from the boundary slightly. The contrastive loss is defined as:

$$\mathcal{L}_t = \sum_{i \in (B \cap \mathcal{L}_n = \mathcal{L}_c)} \mathcal{L}_t^i \tag{4}$$

where $\mathcal{L}_t^i$ is defined as:

$$\mathcal{L}_t^i = -\log \frac{\sum\limits_{j \in N_k(i)} \exp \frac{z_i z_j}{||z_i|| ||z_j||}}{\sum\limits_{m=1}^{k} \exp \frac{z_i z_m}{||z_i|| ||z_j||}} \tag{5}$$

By combining CELoss, $\mathcal{L}_c$ and $\mathcal{L}_t$, the loss function should prevent the model from overfitting and help the model to cluster. The final loss function is defined as:

$$\mathcal{L}_3 = \mathcal{L}_{\text{CELoss}} + \lambda(\mathcal{L}_t + \mathcal{L}_c) \tag{6}$$

In conclusion, we use L_3 to train the network. What's more, the contrastive loss with the characteristics of plug and play can be used in different networks combining with feature extractor and classify head.

3.4. WDBA

In [22], WDBA is designed for time series classification, where the space in which they are embedded is induced by Dynamic Time Warping (DTW). The main idea of our approach is to average a set of time series and use the average time series as a new synthetic example. Using the datasets generated from the approach can reduce the noise and keep the original features. We find that if we can use the WDBA in both the training set and the test set with their true labels we can have a very good outcome to classify. However, we can not use the test set label. We also experiment that if we just have partly true labels in the test set it is also helpful to make the classification. The process and the detail will show in 4.2. In 3.3, we can get the pseudo label and if we can use the label in WDBA, we called PWDBA(Pseudo Weighted DTW Barycentric Averaging), which can help the datasets to reduce the noise. After some epochs, we use the original data and the true labels in the train set and the pseudo label in the test set to WDBA. Using the new data to package the data loader continues to train.

Architectural Design

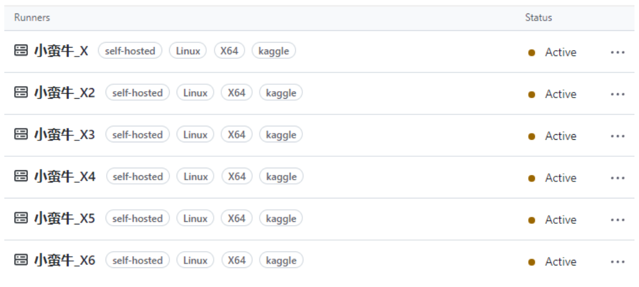
We used Pytorch Lightning, Hydra, Wandb, and Github Integrations to build a stable and clean architecture that allows us to rapidly iterate over new models, datasets, and tasks on different hardware accelerators like CPUs, multi-GPUs or TPUs, even deploy tasks on totally separate machines and collect all experiment metrics, models, training logs. By applying a collection of best practices for efficient workflow and reproducibility, we can stably produce convincing results with Github Action Task deploy the system.

3.5. Main Technologies

- PyTorch Lightning [26] - a lightweight PyTorch wrapper for high-performance AI research. A framework for organizing your PyTorch code.
- Hydra [27] - a framework for elegantly configuring complex applications. The key feature is the ability to dynamically create a hierarchical configuration by composition and override it through config files and the command line.
- Github Action - a series to deploy your established workflows. Using this service, we can automate our workflows, such as testing code quality and feasibility in every push or pull request and on every dataset, or we can easily establish experiments on every model on every dataset and every fold.
- wandb [28] - Weights & Biases is a lightweighted framework. By using it, we can quickly track experiments, version and iterate on datasets, evaluate model performance, reproduce models, visualize results and spot regressions, and share findings with colleagues.

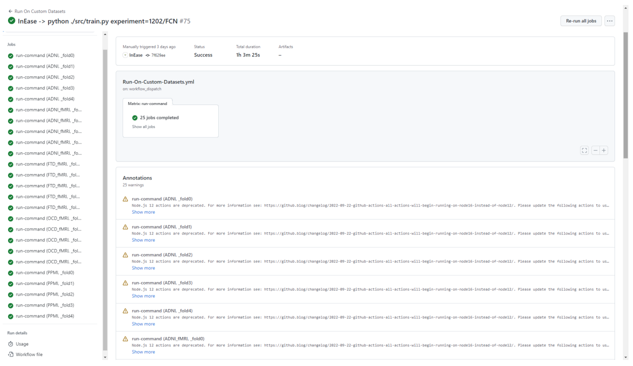
3.6. Main Idea of our design

- **Compact Structure**: clean and scalable so that work can easily be extended



Runners					Status
小蛮牛_X	self-hosted	Linux	X64	kaggle	Active
小蛮牛_X2	self-hosted	Linux	X64	kaggle	Active
小蛮牛_X3	self-hosted	Linux	X64	kaggle	Active
小蛮牛_X4	self-hosted	Linux	X64	kaggle	Active
小蛮牛_X5	self-hosted	Linux	X64	kaggle	Active
小蛮牛_X6	self-hosted	Linux	X64	kaggle	Active

Figure 3. Github Action Self-Hosted Runners



← Run On Custom Datasets

InEase -> python ./src/train.py experiment=1202/FCN #75

Jobs
- run-command (ADNI, _fold0)
- run-command (ADNI, _fold1)
- run-command (ADNI, _fold2)
- run-command (ADNI, _fold3)
- run-command (ADNI, _fold4)
- run-command (ADNI_fMRI, _fo...
- run-command (ADNI_fMRI, _fo...
- run-command (ADNI_fMRI, _fo...
- run-command (ADNI_fMRI, _fo...
- run-command (ADNI_fMRI, _fo...
- run-command (FTD_fMRI, _fol...
- run-command (FTD_fMRI, _fol...
- run-command (FTD_fMRI, _fol...
- run-command (FTD_fMRI, _fol...
- run-command (FTD_fMRI, _fol...
- run-command (OCD_fMRI, _fol...
- run-command (OCD_fMRI, _fol...
- run-command (OCD_fMRI, _fol...
- run-command (OCD_fMRI, _fol...
- run-command (OCD_fMRI, _fol...
- run-command (PPMI, _fold0)
- run-command (PPMI, _fold1)
- run-command (PPMI, _fold2)
- run-command (PPMI, _fold3)
- run-command (PPMI, _fold4)

Run details
Usage
Workflow file

Manually triggered 3 days ago	Status	Total duration	Artifacts
InEase -o- 7f629ee	Success	1h 3m 25s	–

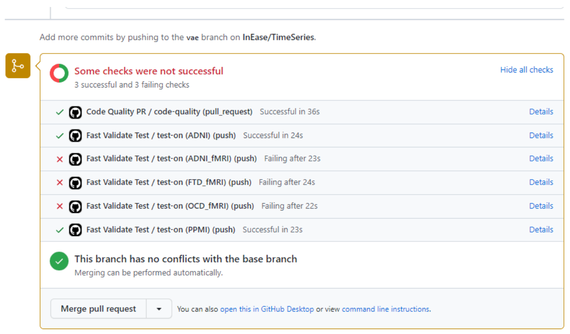
Run-On-Custom-Datasets.yml
on: workflow_dispatch

Matrix: run-command
25 jobs completed
Show all jobs

Annotations
25 warnings

```
run-command (ADNI, _fold0)
Node.js 12 actions are deprecated. For more information see: https://github.blog/changelog/2022-09-22-github-actions-all-actions-will-begin-running-on-node16-instead-of-node12/. Please update the following actions to us...
Show more
run-command (ADNI, _fold1)
Node.js 12 actions are deprecated. For more information see: https://github.blog/changelog/2022-09-22-github-actions-all-actions-will-begin-running-on-node16-instead-of-node12/. Please update the following actions to us...
Show more
run-command (ADNI, _fold2)
Node.js 12 actions are deprecated. For more information see: https://github.blog/changelog/2022-09-22-github-actions-all-actions-will-begin-running-on-node16-instead-of-node12/. Please update the following actions to us...
Show more
run-command (ADNI, _fold3)
Node.js 12 actions are deprecated. For more information see: https://github.blog/changelog/2022-09-22-github-actions-all-actions-will-begin-running-on-node16-instead-of-node12/. Please update the following actions to us...
Show more
run-command (ADNI, _fold4)
Node.js 12 actions are deprecated. For more information see: https://github.blog/changelog/2022-09-22-github-actions-all-actions-will-begin-running-on-node16-instead-of-node12/. Please update the following actions to us...
Show more
run-command (ADNI_fMRI, _fold0)
Node.js 12 actions are deprecated. For more information see: https://github.blog/changelog/2022-09-22-github-actions-all-actions-will-begin-running-on-node16-instead-of-node12/. Please update the following actions to us...
Show more
```

Figure 4. Github Action Matrix include all dataset of all folds



Add more commits by pushing to the vae branch on InEase/TimeSeries.

Some checks were not successful
3 successful and 3 failing checks

- ✓ Code Quality PR / code-quality (pull_request) Successful in 36s — Details
- ✓ Fast Validate Test / test-on (ADNI) (push) Successful in 24s — Details
- ✗ Fast Validate Test / test-on (ADNI_fMRI) (push) Failing after 23s — Details
- ✗ Fast Validate Test / test-on (FTD_fMRI) (push) Failing after 24s — Details
- ✗ Fast Validate Test / test-on (OCD_fMRI) (push) Failing after 22s — Details
- ✓ Fast Validate Test / test-on (PPMI) (push) Successful in 23s — Details

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

Figure 5. Auto test on 5 datasets

- **Rapid Experimentation**: thanks to hydra command line superpowers and well-written configuration files, we can trigger experiments singly. For example, we can easily toggle debug mode by:

```
python train.py debug=default
```

- **Easily distributed**: By using GitHub Action Self-Hosted Servers, we can automatically deploy our machine learning tasks to different machines. Here all machines are started from the Kaggle CPU machine (see fig.3), and by using them, we can run up to 12 experiments at a time (see fig.4).
- **Main Configs**: auto load default training configuration

and .env file, everyone can specify different configurations on their local machine and not affect others.

- **Experiment Configs**: override chosen hyperparameters to perform fast staging experiments.
- **Clean and diced codes:**Legibility is our number one pursuit since we need to collaborate with our teammates and explain our code to others, so we will perform strict code quality checks on every commit, avoiding strange code styles.
- **Logs**: all logs (checkpoints, configs, etc.) are stored in a dynamically generated folder structure, and with wand Integration, we can easily compare between different experiments
- **Hyperparameter Search**: made easier with Hydra plugins like Optuna Sweeper
- **Continuous Integration**: automatically test our repo with Github Actions. For Example, in the picture below, the value model in failing on three datasets (see fig.5), which indicates that there may be something wrong in our code and we can inspect it quickly.

4. Experiment Results

4.1. Experiment Configurations

Out proposed models and compared baseline have been tested on all UCR Time Series Datasets. The FCN block configuration and lstm configuration was kept constant in all our experiments, and tuning hyper parameters may increase the performance but we didn‘t do that. The number of training epochs was generally between 200 epochs and 300 epochs, and according to our experiment results, 100 epochs is enough, and since we have set up early stopping, the training process usually stops at 120 epoch. The learning rate of our LSTMFCNPlus is 1e-4, and we set k_neighbor_rate1 to 0.5, k_neighbor_rate2 to 0.3, lambda to 0.5 as is shown in the configuration file. For FCN and MLP, we set the learning rate a little higher to 1e-3. We used Adam optimizer in all our experiments, and set the dropout rate to 80%. Class imbalance is handled by BorderlineSMOTE.

4.2. Time Series Dataset Description

The ADNI_90_120 dataset consists of 48 NC (normal sample), 59 AD (patient, Alzheimer's disease); 56 EMCI, and 43 LMCI (EMCI, meaning early MCI, and LMCI, meaning late MCI symptoms). MCI is an intermediate state between NC and AD. Each sample is a 90*120 data structure. The meaning is the value of 90 brain regions in 120 seconds (blood oxygen concentration) The OCD dataset contains 20 normal samples (NC), 62 patients (OCD). Each sample is a 90*200 type of data. The meaning is the change of blood oxygen levels in 90 brain regions over 200 seconds. The FTD dataset contains 86 normal samples (NC) and 95 patients (FTD). The meaning of each sample is the same as OCD. The PPMI dataset contains 169 normal samples (NC) and 374 patients (PD). Each sample contains 3*98=294 columns of features. Each 98-column feature meaning is CSF, GM, WM (gray matter protein, etc.) in order. The ADNI dataset contains 52 normal samples (NC), 99 MCI (intermediate state, specifically 56 MCIn and 43 MCIp) and 51 patients (AD). The first 1-93 columns of each sample are MRI and the last 94-186 columns are PET.

4.3. Results on Our Datasets

In project1, we use the LSTM, MLP, FCN as a network architecture to train the rs-fMRI datasets and use MLP, FCN, SVM and Extreme Random Tree as a network architecture to train the other datasets. Now, we just use some data prepocessing strategies to make the improvement. Table 1,2 shows the last outcome and the new outcome. We can see that if we can use a good data preprocessing method, the outcomes are not-bad.

In project 2, we use the LSTMFCN as the network architecture and have the experiment. With the Squeeze-and-Excitation block and the loss we design, we can get a better outcome. Also, we think the contrastive loss has the characteristics of plug and play. We use it in the FCN. It shows that it can also get good accuracy. Table 3 shows the outcome.

5. Ablation Experiments

5.1. Cluster Loss

During we use the original LSTMFCN to train just using the cross-entropy loss, we find that it easily has the problem of overfitting. We want to use a way to let the data have the same label cluster and leave far away from the data with a different label. If we can let the clusterings leave far enough, we can effectively reduce overfitting. So we design the clustering loss and use it to train with the train set. The fig.6 is the distribution at the beginning of training.

After training some epochs, the fig.7 shows the traditional LSTMFCN, and the fig.8 shows the LSTMFCN using the clustering loss. We can see the clustering effect in fig.8 which can help the model to train the classification.

However, we can also see some test set data also in the boundary so it may not classify precisely. Also, after some epochs, the test set data will leave in the boundary.

So we want to use the test set feature to constraint and help the test set to cluster. We design the contrastive loss, we can find that after a few epochs the clusterings of the train set and the test set are pretty well. So we use the neighbor samples label and the output of the classifier to constrain the pseudo-label, and we use the contrastive loss to cluster in the test set. And using this loss, we can easily see the effect.

datasets	MLP	FCN	LSTM
OCD	84.11(79.22)	84.24(82.01)	81.57(80.52)
FTD	66.27(65.73)	70.91(72.95)	64.84(65.51)
ADNI_90_120	52.42(55.15)	57.55(56.34)	56.39(50.42)

Table 1. Project1 Performances

datasets	MLP	SVM	Extreme Random Tree	FCN
PPMI	72.21(71.46)	68.83(68.83)	70.72(68.21)	**75.51**(73.85)
ADNI	83.77(73.42)	65.44(65.78)	68.11(68.44)	**91.97**(86.71)

Table 2. Project1 Performances

datasets	LSTMFCN(proposed)	FCN+loss	LSTMFCN w/o Lc
OCD	**87.02**	86.76	85.33
FTD	**78.32**	76.68	75.91
ADNI_90_120	**63.54**	56.94	55.57
PPMI	**77.62**	75.24	75.02
ADNI	58.66	**60.33**	56.28

Table 3. Project2 Performances

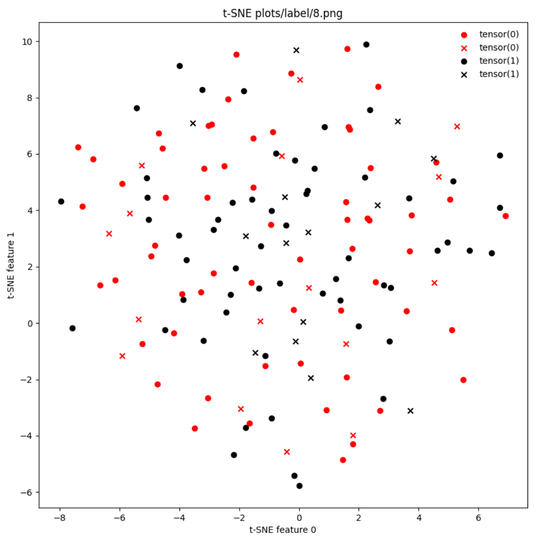


Figure 6. Distribution of Data (at the beginning of train)

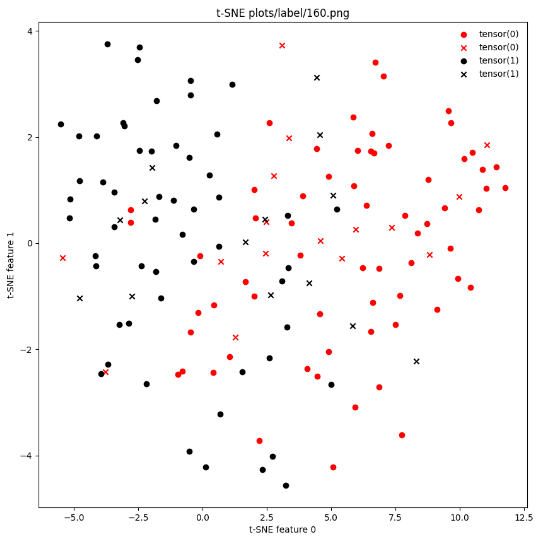


Figure 7. Distribution of Data (after train, LSTM-FCN)

fig.9 is the model with only cross-entropy loss and fig.10 is with just clustering loss. But the fig.11 is with the constraint loss and it shows that the test set can be left far from the boundary.

5.2. WDBA

We also try the different data preprocessing ways, we use t-SNE to visualize the distribution of the datasets to evaluate these approaches. If we don't use any data preprocessing strategies. The distribution of data is shown in fig.12a. we can see they are Stacked together. When we use wdba in both training set and the test set with true labels. The char-

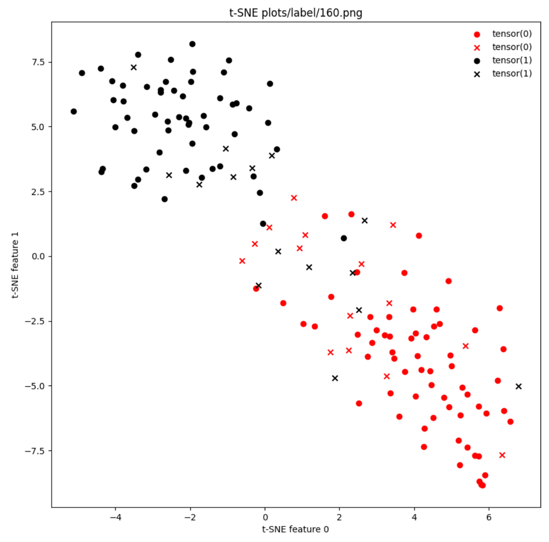


Figure 8. Distribution of Data (after train, LSTM-FCN with ClusterLoss)

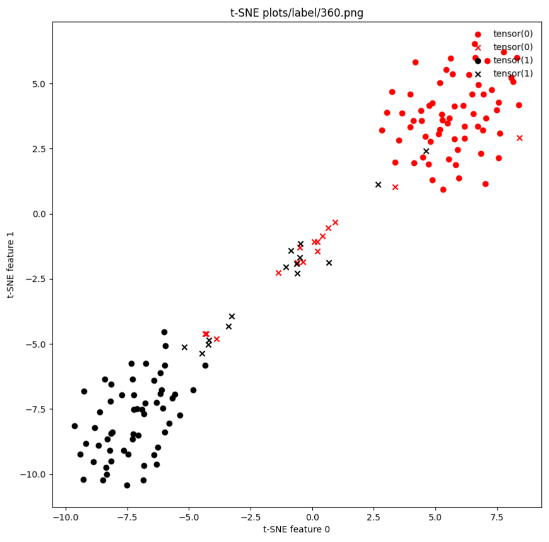


Figure 10. Distribution of Data (after train, Only Cluster Loss)

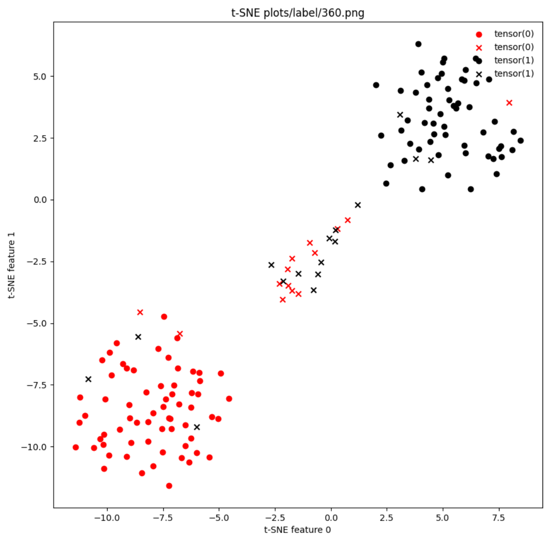


Figure 9. Distribution of Data (after train, Only CE Loss)

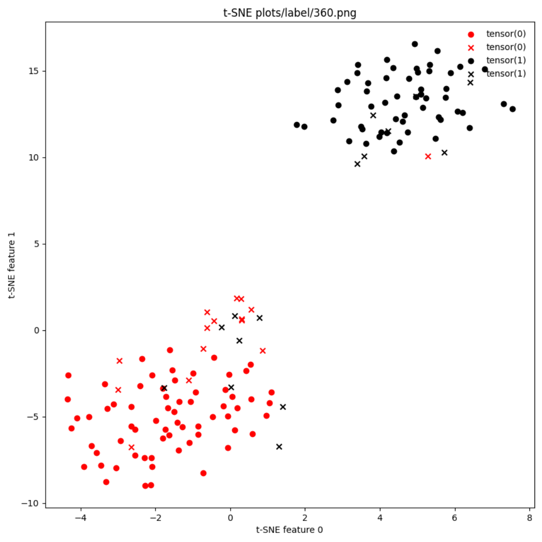


Figure 11. Distribution of Data (after train, with Contrastive Loss)

acteristics of clustering are obviously in fig.12b. With scale, we can get fig.12c. So if we use the approach even with pseudo labels having some mistakes, we can also get a better outcome.

6. Conclusion

In this paper, we use the scale to make the data preprocessing and introduce the LSTMFCN as the network to make the training. In training, we design a clustering loss and a contrastive loss with the characteristics of plug and play to aid training. During the process, we can get the pseudo labels of the test set and we use them to PWDBA which can make

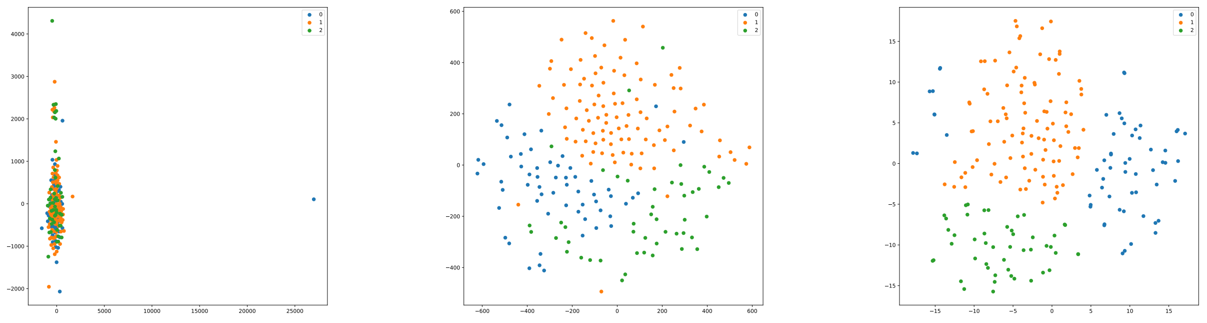

(a) Original Data (b) Data with wdba (c) Data with wdba and scale

Figure 12. ADNI_fMRI tSNE visualize

the training stabler.

The contrastive loss with the characteristics of plug and play can also improve the other model. Surely, the outcomes are also improved by using it.

6.1. Discussion and Future work

We use the contrastive loss to constrain the classification on the test set. And we just use the L2 distance to measure the k-neighbors and use cosine similarity to measure the similarity between the sample and k-neighbors. But we can also test different ways to measure especially DTW due to the advantage to measure the difference between time series. At the same time, the PWDBA can be used to reduce the noise and make the training stabler. However, the pseudo label we use in the method is decided by the classification and clustering in the train set. We can not have an effective way to improve the quality of the pseudo label and we just let it have a stable performance but if we can get a good pseudo label, we can improve the accuracy obviously and vice versa. The datasets are about medical images and are full of lots of medical information. If we can have good enough prior knowledge which can make better pseudo labels, it can make a big improvement.

References

[1] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015. 1

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016. 1

[3] A. P. Ruiz, M. Flynn, J. Large, M. Middlehurst, and A. Bagnall, "The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 35, no. 2, pp. 401–449, 2021. 1

[4] J. Gan, Z. Peng, X. Zhu, R. Hu, J. Ma, and G. Wu, "Brain functional connectivity analysis based on multi-graph fusion," *Medical image analysis*, vol. 71, p. 102057, 2021. 1

[5] F. Karim, S. Majumdar, H. Darabi, and S. Chen, "Lstm fully convolutional networks for time series classification," *IEEE access*, vol. 6, pp. 1662–1669, 2017. 1, 2

[6] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," 2017. 1, 2, 3

[7] C. Orsenigo and C. Vercellis, "Combining discrete svm and fixed cardinality warping distances for multivariate time series classification," *Pattern Recognition*, vol. 43, no. 11, pp. 3787–3794, 2010. 2

[8] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification. acm sigkdd explor. newsl. 12 (1), 40–48 (2010)." 2

[9] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," in *International conference on web-age information management*, pp. 298–310, Springer, 2014. 2

[10] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," 2016. 2

[11] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," 2016. 2

[12] X. Chen, H. Fan, R. Girshick, and K. He, "Improved baselines with momentum contrastive learning," 2020. 2

[13] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, pp. 1597–1607, PMLR, 2020. 2

[14] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18661–18673, 2020. 2

[15] C. Liu, Y. Fu, C. Xu, S. Yang, J. Li, C. Wang, and L. Zhang, "Learning a few-shot embedding model with contrastive learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 8635–8643, 2021. 2

[16] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu, "Time series data augmentation for deep learning: A survey," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, International Joint Conferences on Artificial Intelligence Organization, aug 2021. 2

[17] B. K. Iwana and S. Uchida, "An empirical survey of data augmentation for time series classification with neural networks," *PLOS ONE*, vol. 16, p. e0254841, jul 2021. 2

[18] T. T. Um, F. M. J. Pfister, D. Pichler, S. Endo, M. Lang, S. Hirche, U. Fietzek, and D. Kulić, "Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks," in *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, ACM, nov 2017. 2

[19] K. M. Rashid and J. Louis, "Window-warping: a time series data augmentation of imu data for construction equipment activity identification," in *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, vol. 36, pp. 651–657, IAARC Publications, 2019. 2

[20] A. Le Guennec, S. Malinowski, and R. Tavenard, "Data augmentation for time series classification using convolutional neural networks," in *ECML/PKDD workshop on advanced analytics and learning on temporal data*, 2016. 2

[21] K. Kamycki, T. Kapuscinski, and M. Oszust, "Data augmentation with suboptimal warping for time-series classification," *Sensors*, vol. 20, no. 1, p. 98, 2019. 2

[22] G. Forestier, F. Petitjean, H. A. Dau, G. I. Webb, and E. Keogh, "Generating synthetic time series to augment sparse datasets," in *2017 IEEE International Conference on Data Mining (ICDM)*, pp. 865–870, 2017. 2, 3, 4

[23] B. K. Iwana and S. Uchida, "Time series data augmentation for neural networks by time warping with a discriminative teacher," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 3558–3565, IEEE, 2021. 2

[24] F. Karim, S. Majumdar, H. Darabi, and S. Chen, "LSTM fully convolutional networks for time series classification," *IEEE Access*, vol. 6, pp. 1662–1669, 2018. 2

[25] F. Karim, S. Majumdar, H. Darabi, and S. Harford, "Multivariate LSTM-FCNs for time series classification," *Neural Networks*, vol. 116, pp. 237–245, aug 2019. 4

[26] W. Falcon and The PyTorch Lightning team, "PyTorch Lightning," 3 2019. 5

[27] O. Yadan, "Hydra - a framework for elegantly configuring complex applications." Github, 2019. 5

[28] L. Biewald, "Experiment tracking with weights and biases," 2020. Software available from wandb.com. 5