Project on:

Adaptive Memory Networks with Self-supervised Learning for Unsupervised Anomaly Detection

By
TANUSHREE GORAI - 20BCE1269
MRIDUL JADON - 20BCE1734
POULAMI BERA - 20NCE1305

Course code & Course Title : CSE4020 - Machine Learning
Faculty : Dr. Syed Ibrahim S P

Project Abstract :

- Unsupervised anomaly detection tries to construct models to effectively detect undetected anomalies by just training on normal data. Although prior reconstruction-based approaches have made significant strides, their capacity to generalise is constrained by two major obstacles. First off, the training dataset is restricted to only normal patterns, which hinders the generalizability of the model. Second, it is frequently difficult to maintain the diversity of normal patterns because the feature representations learnt by previous models frequently lack representativeness. To overcome these difficulties and improve the generalizability of unsupervised anomaly detection, we suggest a method termed Adaptive Memory Network with Self-supervised Learning. The AMSL comprises a self-supervised learning module to learn generic normal patterns based on the convolutional auto encoder structure and an adaptive memory fusion module to learn rich feature representations.

3 main point

- Self-supervised learning and adaptive memory fusion are applied to compensate for the diversity in normal time series data and the information that cannot be obtained from limited training data.
- The model is fast and has little performance degradation even when it is lightweight.
- Deep insight into the behavior of data (signals) is important to increase the accuracy of the model.

Two major challenges :

- Lack of normal data: Lack of normal data but there are many variations and expansions of normal data, and we cannot prepare a data set that includes all of them for training. (a) normal, (b) abnormal, and (c) similar.
- Limitation of feature representation: When there is diversity in normal data like (c), conventional methods cannot represent it well.

The aim of the Adaptive Memory Network with Self-supervised Learning (AMSL) proposed in this paper is as follows :

- Self-supervised learning and memory networks for normal data and feature representation tasks, respectively.
- Learning global and local memories to increase expressive ability, and then using the adaptive memory fusion module to fuse global and local memories into a final expression.
- We compare the performance of four public data sets. Compared to the conventional method, we observed more than a 4% improvement in accuracy and F1 score. It is also more robust to noise.

Related technology

- Unsupervised anomaly detection in deep learning methods can be categorized into reconstruction models and prediction models.

Reconstruction model

We focus on reducing the reconstruction error. For example, autoencoders are often used for anomaly detection by learning to reconstruct a given input, while LSTM encoder decoders are used for time-series data but cannot account for spatial correlation. For time-series data, LSTM encoder-decoders are used, but they cannot take into account spatial correlations; Convolutional autoencoders can capture 2D image structures; ConvolutionalLSTM can capture spatial and temporal correlations.

Prediction model

It predicts one or more consecutive values. For example, RNN-based models detect anomalies based on the error between future predictions and actual values; LSTNet captures short- and long-term patterns; GAN-based methods use U-Net as a generator to predict the next point in time and compare it to actual values to detect anomalies, and GAN-based methods use U-Net as a generator to predict the next point in time and compare it to actual values to detect anomalies. However, these methods lack a reliable mechanism for the granular representation of normal data.

Self-supervised learning

- Feature representation learning is one important aspect of deep learning, where a good representation of the input data is essential for generalizability, interpretability, and robustness. Self-supervised learning (SSL) is one of the unsupervised learning paradigms that use the data itself to obtain a good representation. Specific methods span image, natural language processing, and speech recognition. In anomaly detection, it is used to learn features of within-distribution (i.e. normal) samples.

Storage network

- It is used for question answering.RNN, LSTM uses local memory cells to understand the long-term structure. Memory records information stably, so we employ memory networks like one-shot learning, neural machine translation, and anomaly detection. Anomaly detection aims to distinguish between normal and abnormal values by recording various patterns of normal values compared to the items in the memory.

Configuration of AMSL

- Convolutional AE (CAE) is used as the base network; the loss function of CAE is the mean squared error (MSE) as shown in Eq.

$$L_{MSE} = \|\boldsymbol{x} - \boldsymbol{x}'\|_2^2, \qquad (2)$$

- AMSL is composed of four elements.

1) Self-supervised learning module
2) Global memory module
3) Local memory module
4) Adaptive fusion module

And the algorithm also consists of four steps.

- The encoder maps the raw time-series data into the latent feature space by performing six transformations.
- For self-supervised learning, a multi-class classifier classifies these feature representations to learn generalized features.
- Features are also sent to global and local storage networks to learn common and individual features.
- An adaptive fusion module fuses these features to obtain a new representation that can be used for the reconstruction.

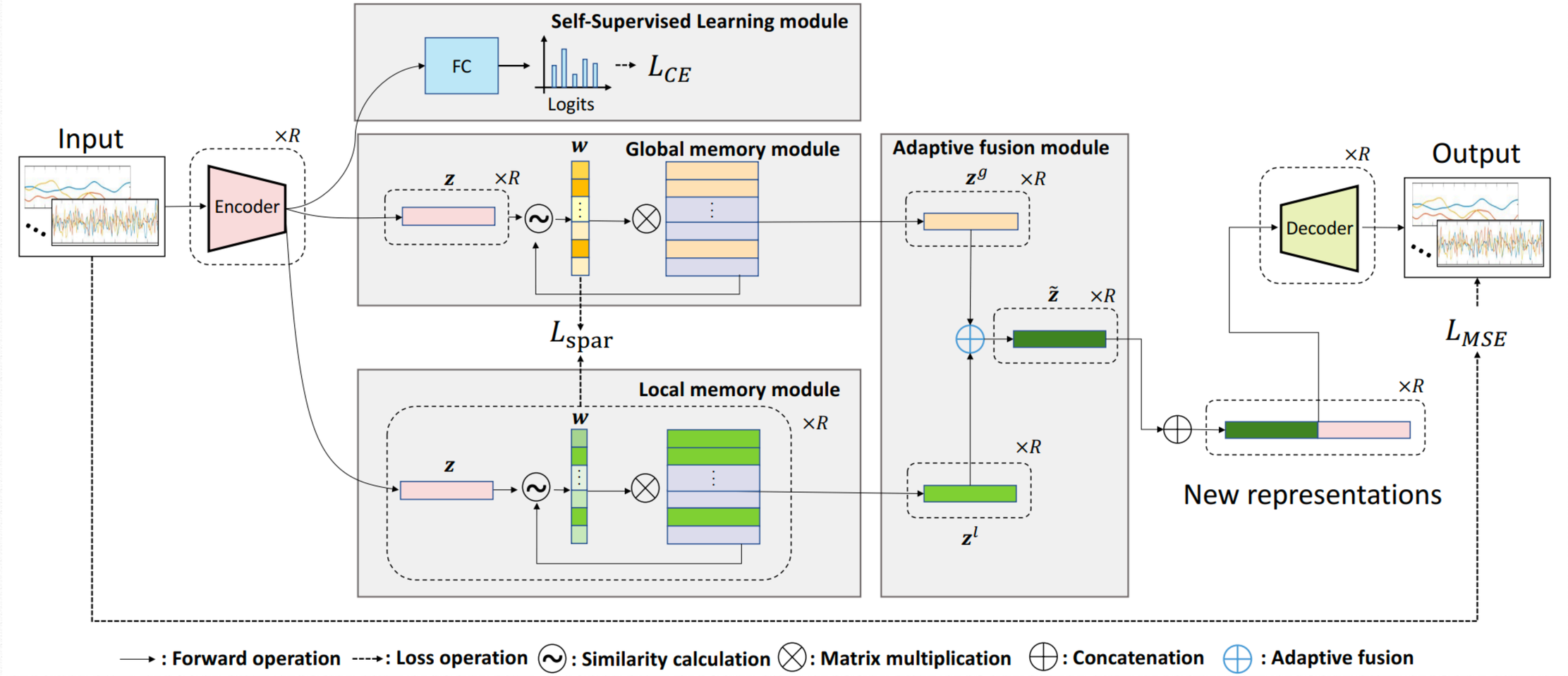


Fig. 2: The structure of the proposed AMSL. It consists of four components: self-supervised learning, global memory, local memory and adaptive fusion. The notation "$\times R$" denotes R copies where each one corresponds to one transformation.

Evaluation test

DSADS is motion sensor data on daily body movements :

- Sitting, standing, lying on back, lying on right side, ascending stairs, descending stairs, standing in an elevator still, moving around in an elevator, walking in a parking lot, walking on a treadmill with a speed of 4 kmh, walking in flat and 15 deg inclined positions, running on a treadmill with a speed of 8 kmh, exercising on a stepper, exercising on a cross trainer, cycling on an exercise bike in horizontal positions, cycling on an exercise bike in vertical positions, rowing, jumping, playing basketball

- Anoamly classes including running, ascending stairs, descending stairs and rope jumping

- TABLE 1 : The detailed statistics of the dataset:

Dataset	Application	#Instance	#Dim	#Class
DSADS [47]	Activity Recognition	1,140,000	45	19

- TABLE 2 : shows the classification of normal and abnormal by operation for DSADS :

(a) DSADS

Activity	Class
sitting, standing, lying, moving, walking, cycling, exercising, rowing	normal
running, ascending stairs, descending stairs, rope jumping, playing basketball	abnormal

TABLE 3: The comparison of mean precision, recall, F1 and accuracy of AMSL and other baselines. The best and second-best results are bold and underlined, respectively. We can see that AMSL significantly outperforms other methods.

Method	DSADS dataset			
	mPre	mRec	mF1	Acc
Kernel PCA [11]	0.6184	0.6182	0.6183	0.6186
ABOD [50]	0.6880	0.6510	0.6690	0.6554
OCSVM [15]	0.7608	0.7277	0.7439	0.7312
HMM [51]	0.6959	0.6917	0.6937	0.6901
CNN-LSTM [52]	0.6845	0.6270	0.6545	0.6425
LSTM-AE [8]	0.8471	0.7729	0.8083	0.7852
MSCRED [9]	0.7540	0.5602	0.6428	0.6192
ConvLSTM-AE [16]	0.8164	0.6951	0.7509	0.7121
ConvLSTM-COMP [16]	0.8229	0.7379	0.7781	0.7518
BeatGAN [53]	0.9517	0.5663	0.7100	0.7818
MNAD-P [43]	0.5816	0.5783	0.5799	0.5721
MNAD-R [43]	0.8337	0.7694	0.8003	0.7811
GDN [54]	0.8706	0.8151	0.8419	0.8251
UODA [23]	<u>0.8679</u>	<u>0.8281</u>	<u>0.8475</u>	<u>0.8365</u>
AMSL (Ours)	**0.9407**	**0.9298**	**0.9352**	**0.9332**
Improvement	7.28%	10.17%	8.77%	9.67%

- The models to be compared are four traditional methods (KPCA, ABOD, OCSVM, HMM) and seven unsupervised learning methods (CNN-LSTM, LSTM AE, MSCRED, BeatGAN, MNAD, GDN, UODA, CovLSTM-COMPOSITE). The evaluation metrics are average fit rate, average recall rate, average F1 score, and accuracy.

- TABLE 3 shows the evaluation results. For the database DSADS, we find that the amount of improvement decreases as the number of data increases. This means that self-supervised learning is more effective when generalized representations are difficult to learn on small data sets. Furthermore, while the number of samples is relatively small, the improvement in AMSL is large when the number of categories is large, indicating its superior ability to handle diversity on limited training data.

- TABLE 3 reports the overall performance results on these public datasets. It can be observed that the proposed AMSL method achieves significantly superior performance over the baseline methods in all the datasets. Specifically, compared with other methods, AMSL significantly improves the F1 score by 8.77% on DSADS dataset. The same pattern goes for precision and recall.

- The confusion matrix in Fig.4 shows that for most of the datasets, the ratio of misclassification of normal data is lower than that of misclassification of abnormal data; the F1 score is over 93%.

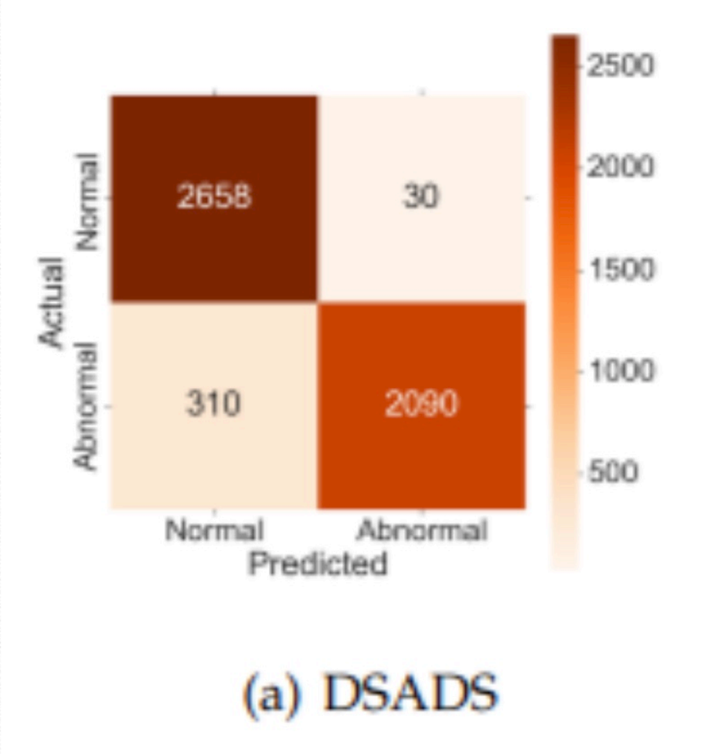
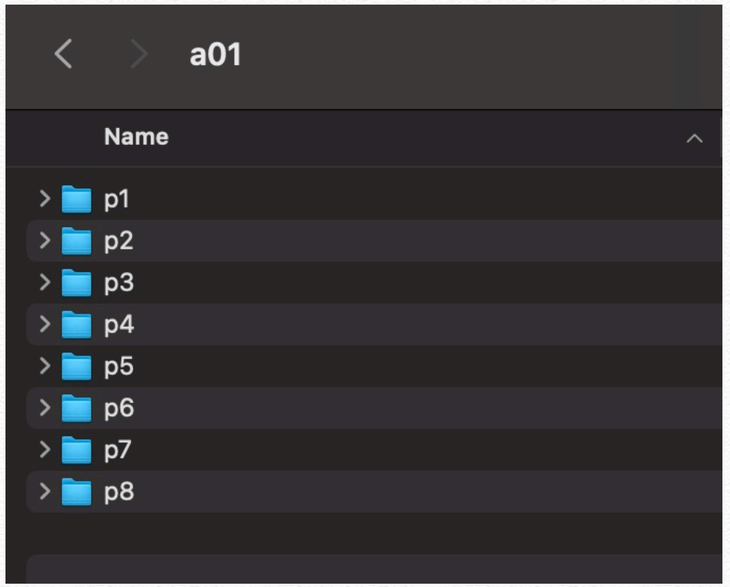
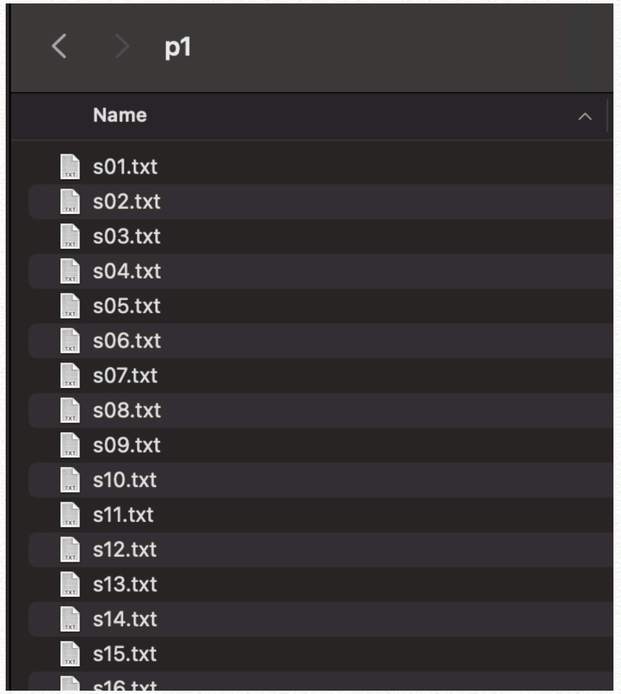


Fig. 4: The confusion matrices on four datasets.

Dataset : DASADS





Data Preparation :

We use DASADS dataset. Daily and Sports Activities Data Set. The dataset comprises motion sensor data of 19 daily and sports activities each performed by 8 subjects in their own style for 5 minutes.

- Run preprocessing.py to generate normal and abnormal datasets.

```
data
|-DASADS
| |-a01
| | |-p1
| | | |-s01.txt
| |-...
| |-a09
|-generate_dataset
| |-normal.npy
| |-abnormal.npy
```

- Run transformation.py to transform all the data and then separate them into training and testing datasets.

```
data
 |-generate_dataset
 | |-normal.npy
 | |-abnormal.npy
 |-transform_dataset
 | |-train_dataset
 | | |-data_raw_train.npy  # raw data
 | | |-data_no_train.npy   # noise data
 | | |-data_ne_train.npy   # negated data
 | | |-data_op_train.npy   # opposite_time data
 | | |-data_pe_train.npy   # permuted data
 | | |-data_sc_train.npy   # scale data
 | | |-data_ti_train.npy   # time_warp data
 | |-test_dataset
 | | |-normal data
 | | | |-data_raw_test.npy  # raw data
 | | | |-data_no_test.npy   # noise data
 | | | |-data_ne_test.npy   # negated data
 | | | |-data_op_test.npy   # opposite_time data
 | | | |-data_pe_test.npy   # permuted data
 | | | |-data_sc_test.npy   # scale data
 | | | |-data_ti_test.npy   # time_warp data
 | | |-abnormal data
 | | | |-data_raw_abnormal.npy  # raw data
 | | | |-data_no_abnormal.npy   # noise data
 | | | |-data_ne_abnormal.npy   # negated data
 | | | |-data_op_abnormal.npy   # opposite_time data
 | | | |-data_pe_abnormal.npy   # permuted data
 | | | |-data_sc_abnormal.npy   # scale data
 | | | |-data_ti_abnormal.npy   # time_warp data
```

Run:

- Train model : You can get results of the MSE loss after running train.py.results

|-train_normal_loss_sum_mse.csv #the MSE loss of training data
|-normal_loss_sum_mse.csv #the MSE loss of normal data in the testing dataset
|-abnormal_loss_sum_mse.csv #the MSE loss of abnormal data in the testing dataset

Evaluation:

- Run evaluate.py to compute the threshold by the MSE loss of training data and achieve the accuracy, precision, recall and F1 score of testing data.

Preprocessing.py

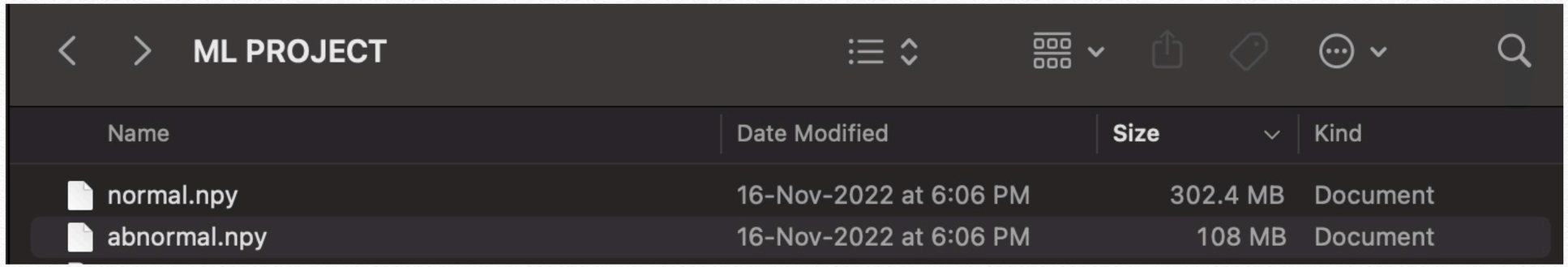
```
import numpy as np
path = '/Users/tanushree/Desktop/ML PROJECT/data/'
ac_class = []
for i in range(1,20):
    if i < 10:
        ac_class.append('a0'+str(i))
    else:
        ac_class.append('a'+str(i))

ac_person = []
for i in range(1,9):
        ac_person.append('p'+str(i))

ac_file = []
for i in range(1,61):
    if i < 10:
        ac_file.append('s0' + str(i))
    else:
        ac_file.append('s' + str(i))

print(ac_class)
print(ac_person)
print(ac_file)
```

```
##################abnormal class#########################
abnormal_class = [4,5,11,17,18]
abnormal = []
for i in range(5):
    for j in range(8):
        for z in range(60):
            print(ac_class[abnormal_class[i]] +'/'+ ac_person[j]+'/'+ ac_file[z]+ '.txt')
            data = np.loadtxt(path + ac_class[abnormal_class[i]] +'/'+
                              ac_person[j]+'/'+ ac_file[z] + '.txt',delimiter= ',')
            abnormal.append(data)
    print(len(abnormal))
abnormal = np.array(abnormal)
print(abnormal.shape)
np.save("/Users/tanushree/Desktop/ML PROJECT/abnormal.npy", abnormal)
```

```
##################normal class#########################
normal = []
normal_class = [i for i in range(19) if i not in [4,5,11,17,18]]
for i in range(14):
    for j in range(8):
        for z in range(60):
            print(ac_class[normal_class[i]] +'/'+ ac_person[j]+'/'+ ac_file[z]+ '.txt')
            data = np.loadtxt(path + ac_class[normal_class[i]] +'/'+
                              ac_person[j]+'/'+ ac_file[z] + '.txt',delimiter= ',')
            normal.append(data)
    print(len(normal))
normal = np.array(normal)
print(normal.shape)
np.save("/Users/tanushree/Desktop/ML PROJECT/normal.npy", normal)
```

Output:

```
['a01', 'a02', 'a03', 'a04', 'a05', 'a06', 'a07', 'a08', 'a09', 'a10', 'a11', 'a12', 'a13', 'a14', 'a15', 'a16', 'a1
7', 'a18', 'a19']
['p1', 'p2', 'p3', 'p4', 'p5', 'p6', 'p7', 'p8']
['s01', 's02', 's03', 's04', 's05', 's06', 's07', 's08', 's09', 's10', 's11', 's12', 's13', 's14', 's15', 's16', 's1
7', 's18', 's19', 's20', 's21', 's22', 's23', 's24', 's25', 's26', 's27', 's28', 's29', 's30', 's31', 's32', 's33',
's34', 's35', 's36', 's37', 's38', 's39', 's40', 's41', 's42', 's43', 's44', 's45', 's46', 's47', 's48', 's49', 's5
0', 's51', 's52', 's53', 's54', 's55', 's56', 's57', 's58', 's59', 's60']
a01/p1/s01.txt
a01/p1/s02.txt
a01/p1/s03.txt
a01/p1/s04.txt
a01/p1/s05.txt
a01/p1/s06.txt
a01/p1/s07.txt
a01/p1/s08.txt
a01/p1/s09.txt
a01/p1/s10.txt
a01/p1/s11.txt
a01/p1/s12.txt
```



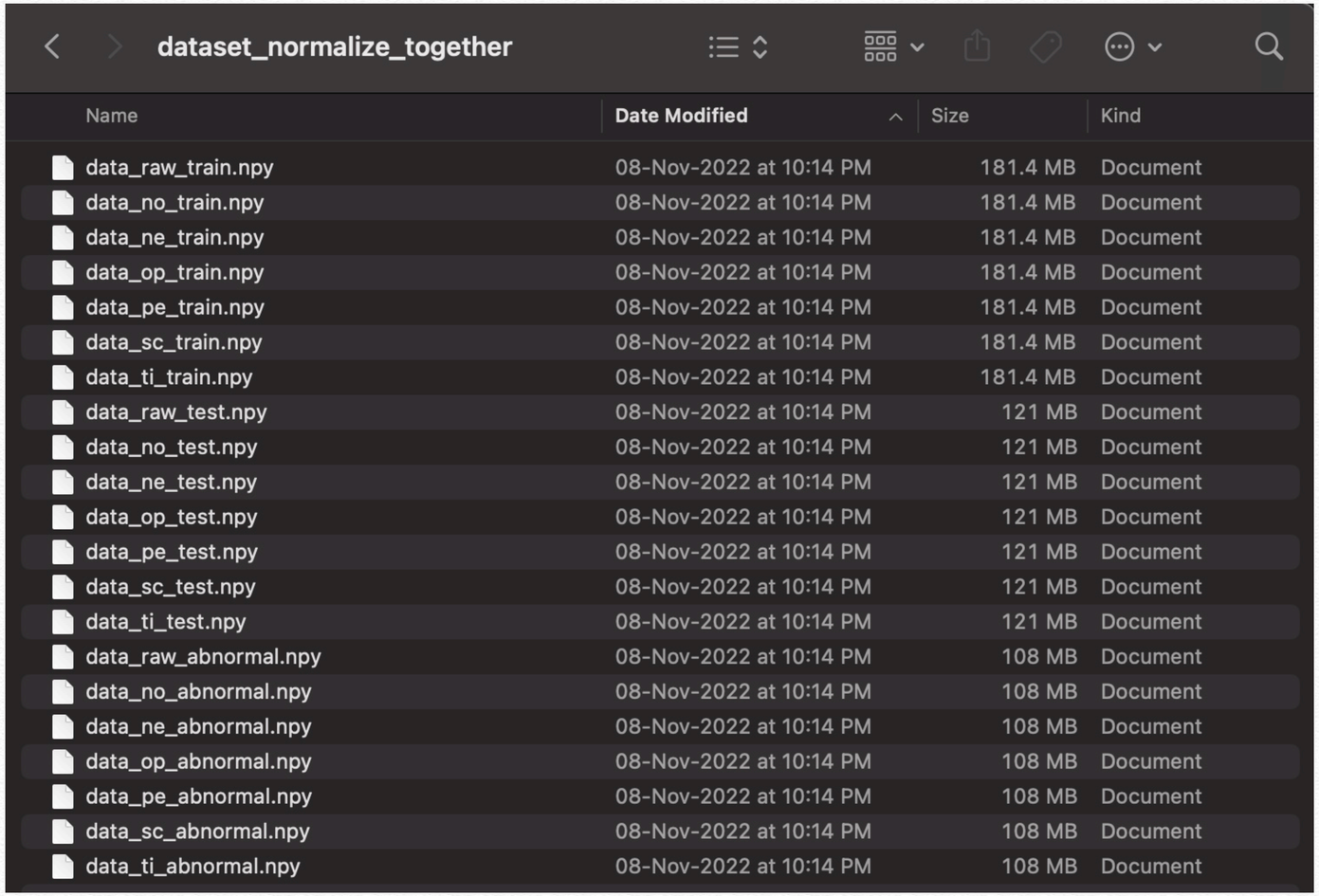
Transformation.py

```
####################shuffle data#######################
listA = [l for l in range(normal_s_no.shape[0])]
random.shuffle(listA)
listB = [p for p in range(normal_s_no.shape[0])]

dataset_raw = np.zeros(
    [normal_s_no.shape[0], normal_s_no.shape[1], normal_s_no.shape[2], normal_s_no.shape[3]])
dataset_no = np.zeros(
    [normal_s_no.shape[0], normal_s_no.shape[1], normal_s_no.shape[2], normal_s_no.shape[3]])
dataset_ne = np.zeros(
    [normal_s_no.shape[0], normal_s_no.shape[1], normal_s_no.shape[2], normal_s_no.shape[3]])
dataset_op = np.zeros(
    [normal_s_no.shape[0], normal_s_no.shape[1], normal_s_no.shape[2], normal_s_no.shape[3]])
dataset_pe = np.zeros(
    [normal_s_no.shape[0], normal_s_no.shape[1], normal_s_no.shape[2], normal_s_no.shape[3]])
dataset_sc = np.zeros(
    [normal_s_no.shape[0], normal_s_no.shape[1], normal_s_no.shape[2], normal_s_no.shape[3]])
dataset_ti = np.zeros(
    [normal_s_no.shape[0], normal_s_no.shape[1], normal_s_no.shape[2], normal_s_no.shape[3]])

for w, r in zip(listA, listB):
    dataset_raw[r, :, :, :] = normal_s_raw[w, :, :, :]
    dataset_no[r, :, :, :] = normal_s_no[w, :, :, :]
    dataset_ne[r, :, :, :] = normal_s_ne[w, :, :, :]
    dataset_op[r, :, :, :] = normal_s_op[w, :, :, :]
    dataset_pe[r, :, :, :] = normal_s_pe[w, :, :, :]
    dataset_sc[r, :, :, :] = normal_s_sc[w, :, :, :]
    dataset_ti[r, :, :, :] = normal_s_ti[w, :, :, :]

print('shuffle done')
```

```
#################################save normal data#####################
shuffle( data_raw_n,data_no_n,data_ne_n,data_op_n,data_pe_n,data_sc_n,data_ti_n)

#################################save abnormal data#####################
path = '/Users/tanushree/Desktop/ML PROJECT/dataset_normalize_together/'
np.save(path + "data_raw_abnormal.npy", data_raw_a)
np.save(path + "data_no_abnormal.npy", data_no_a)
np.save(path + "data_ne_abnormal.npy", data_ne_a)
np.save(path + "data_op_abnormal.npy", data_op_a)
np.save(path + "data_pe_abnormal.npy", data_pe_a)
np.save(path + "data_sc_abnormal.npy", data_sc_a)
np.save(path + "data_ti_abnormal.npy", data_ti_a)
```

Output:

```
shuffle done
save preparing
```



Summary

- In this project, we propose an adaptive memory network with self-supervised learning (AMSL) for unsupervised anomaly detection of multivariate time series signals. To enhance the generalization capability of the model for invisible anomalies, we proposed to use a self-supervised learning module to learn a variety of normal patterns and an adaptive memory fusion network to learn rich feature representations by global and local memory modules. Experiments on four public datasets show that AMSL significantly outperforms existing approaches in terms of accuracy, generalization, and robustness.

- In the future, they plan to extend AMSL to other modalities, such as image and video, for unsupervised anomaly detection, and they also plan to develop more efficient learning algorithms and pursue the theoretical analysis of the method.