上海人工智能研究院T2I任务-intern

项目要求：

关于多维时间序列到2D图像相关技术研究

1. **背景**

随着深度学习的不断发展（尤其是在CV和NLP领域），其相关技术也在逐步转换至分析多维时间序列中。目前，主流的时序分析包括RNN以及CNN等，但是由于RNN较难训练，从而导致算法难以应用，而且没有现存与训练网络，并且1D-CNN也不便利。

2. **目标**
3. 调研3种以上encoding方式，并形成相应的文档（包含原理和出处）；
4. 形成相应的code并提交至指定仓库，使用python作为编程语言
5. 在两个公开时序数据集上进行测试，生成的图片可以进行简单聚类
6. **方法**

目前可以参考的方法：递归图，马尔可夫变迁场和格拉米角场

个人Todolist

✍️

1.调研过后弄清楚原理（如为什么要使用，各个方法的优缺点）

交付文档或者PPT 7.8日（含当日）之前完成

2.找到公开数据集和公开代码，及进行测试 .....交付对比结果/代码仓库

交付文档或者PPT 7.8日（含当日）之前完成

3.对于没有的代码进行补充 .....手动实现代码

暂时未定

个人理解核心要义：

🐵

即为第i时间和第j时间上的数值会产生一个对应关系，从而构成一个time*time的图

要把Xi，Xj的图进行映射从而让Xi，Xj 是为了保证转换过后的图能够和原序列有相互可逆的关系

例如假如时间序列上的值直接相加减，也可以构图，但是不具备可逆性

例如

递归图法：

$$R_{i,j} = \Theta(\varepsilon - \|\vec{x}_i - \vec{x}_j\|), \quad \forall i, j \in \{1, \ldots, n - (m-1)\tau\}$$

格拉米角场

$$\begin{aligned} GASF &= [\cos(\phi_i + \phi_j)] && (4) \\ &= \tilde{X}' \cdot \tilde{X} - \sqrt{I - \tilde{X}^2}' \cdot \sqrt{I - \tilde{X}^2} && (5) \\ GADF &= [\sin(\phi_i - \phi_j)] && (6) \\ &= \sqrt{I - \tilde{X}^2}' \cdot \tilde{X} - \tilde{X}' \cdot \sqrt{I - \tilde{X}^2} && (7) \end{aligned}$$

马尔科夫法：

将数据分段

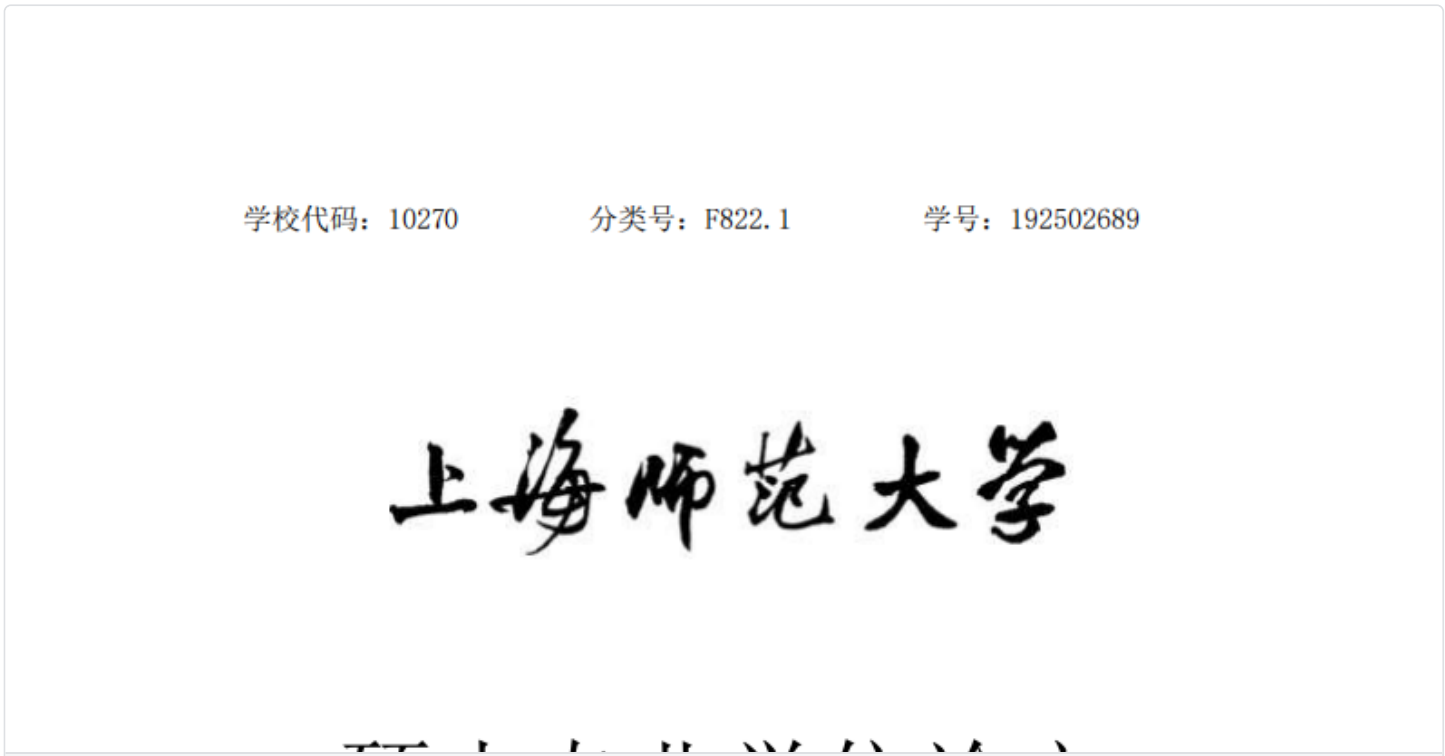
统计时间序列中数据为第i段跳转第j段的频次

最后归一化保证行为1，凑出马尔科夫矩阵（个人想法其实这个地方可以用softmax对行做归一，非线性和线性表示数据大小的敏感程度不同）

$$M = \begin{bmatrix} w_{ij}|x_1 \in q_i, x_1 \in q_j & \cdots & w_{ij}|x_1 \in q_i, x_n \in q_j \\ w_{ij}|x_2 \in q_i, x_1 \in q_j & \cdots & w_{ij}|x_2 \in q_i, x_n \in q_j \\ \vdots & \ddots & \vdots \\ w_{ij}|x_n \in q_i, x_1 \in q_j & \cdots & w_{ij}|x_n \in q_i, x_n \in q_j \end{bmatrix} \quad (8)$$

马尔可夫场缺点分析：

马尔可夫转换场对时间序列的编码方式相对于格莱姆角场的编码方式有一个明显的缺点：在用马尔可夫转换场对时间序列数据进行编码之前，要选定状态的个数，这个操作具有随意性，而且每一个时间段包含的状态个数也是不同的，如果强行将两段时间序列数据的马尔可夫状态规定成一样是不合理的。


学校代码：10270　　分类号：F822.1　　学号：192502689

上海师范大学

基于GAF-CNN对在离岸人民币价差走势的判别及相关研究.pdf

论文阅读笔记

论文阅读小组：

ReadPaper论文阅读平台

对于论文笔记和理解会补充到小组论文平台上

参考代码连接

参考时序代码：

核心仓库：https://github.com/johannfaouzi/pyts

GAF代码：https://github.com/strongnine/Series2Image 即为GASF和GADF

注：S表示Summation ,表示Difference

GAF_MTF_RP_STFT的代码:https://github.com/v3551G/Imaging_sequence_data/tree/master/GAFs_MTF_RP_STFT 这个仓库的代码则是从头造轮子写的如果对算法要改进 可以参考这个，如果是要调研算法可以直接调库，如下图示例代码所示

示例代码

示例代码：GASF和GADF

```
#!/usr/bin/python
#copyright(c) strongnine
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import image
from pyts.image import GramianAngularField

sin_data = np.loadtxt('./data/sinx.csv', delimiter=",", skiprows=0).reshape(1, -1)  # 加载数据 (load the source data)
image_size = 28  # 生成的 GAF 图片的大小 (the size of each GAF image)

# `method` 的可选参数有: `summation` and `difference`
# The optional parameters of argument `method`: `summation` and `difference`
gasf = GramianAngularField(image_size=image_size, method='summation')
sin_gasf = gasf.fit_transform(sin_data)

gadf = GramianAngularField(image_size=image_size, method='difference')
sin_gadf = gadf.fit_transform(sin_data)
imges = [sin_gasf[0], sin_gadf[0]]
titles = ['Summation', 'Difference']

# 两种方法的可视化差异对比
# Comparison of two different methods
fig, axs = plt.subplots(1, 2, constrained_layout=True)
for img, title, ax in zip(imges, titles, axs):
    ax.imshow(img)
    ax.set_title(title)
fig.suptitle('GramianAngularField', y=0.94, fontsize=16)
plt.margins(0, 0)
plt.savefig("./GramianAngularField.pdf", pad_inches=0)
plt.show()

image.imsave("./images/GAF_of_Sin.png", sin_gasf[0])  # 保存图片 (save image)
np.savetxt("./images/GAF_of_Sin.csv", sin_gasf[0], delimiter=',')  # 保存数据为 csv 文件
Footer


```

马尔科夫

```
from pyts.datasets
import load_gunpoint
from pyts.image
import MarkovTransitionField
X, _, _, _ = load_gunpoint(return_X_y=True)
transformer = MarkovTransitionField()
X_new = transformer.transform(X)
X_new.shape
```

递归图

```
from pyts.datasets
import load_gunpoint
from pyts.image
import RecurrencePlot
X, _, _, _ = load_gunpoint(return_X_y=True)
transformer = RecurrencePlot()
X_new = transformer.transform(X)
```

数据集链接：http://www.cs.ucr.edu/~eamonn/time_series_data/

文献部分

[1]《Imaging Time Series to Improve Classification and Imputation》https://readpaper.com/pdf-annotate/note?pdfId=4544600468784570369&noteId=704490097080012800

(1)第一篇阅读笔记试水

《Imaging Time Series to Improve Classification and Imputation》

为什么要将时间序列转为图?

（1）能够将CV的一些Trick用到图里面

（2）对于长序列RNN难训练?

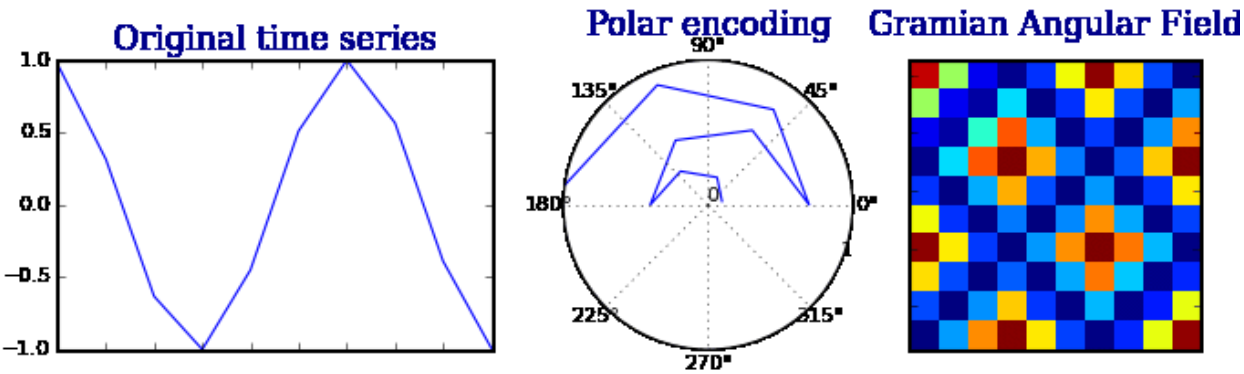
变换的方法：

1.数值归一化，用cos表示，用和 和 差 来表示 不同时间的对应关系

2.马尔可夫方法，计算由i值跳到j值的次数，归一化生成马尔可夫权重矩阵

Tiled卷积的运用

关于可解释性方法?



https://img-blog.csdnimg.cn/2019011313333637.gif

GASF 和 GADF 方法

这篇文章《Imaging Time Series to Improve Classification and Imputation》介绍了如何把时间序列转换成图像，包括 GASF 方法和 GADF 方法。

假设时间序列是 $X = \{x_1, \cdots, x_n\}$ ，长度是 n ，我们可以使用归一化方法把时间序列压缩到 $[0,1]$ 或者 $[-1,1]$ ：

$$\tilde{x}_0^i = (x_i - \min(X))/(\max(X) - \min(X)),$$

$$\tilde{x}_{-1}^i = ((x_i - \max(X)) + (x_i - \min(X)))/(\max(X) - \min(X)),$$

此时的 $\tilde{x}_0^i \in [0,1], \forall 1 \le i \le n$ ， $\tilde{x}_{-1}^i \in [-1,1], \forall 1 \le i \le n$ 。于是可以使用三角函数来代替归一化之后的值。下面通用 $\tilde{x}_i$ 来表示归一化之后的时间序列，令 $\phi_i = \arccos(\tilde{x}_i)$ ，$\tilde{x}_i \in [-1,1]$ ， $1 \le i \le n$ 。因此， $\phi_i \in [0, \pi]$ ，于是， $\sin(\phi_i) \ge 0$ 。

定义矩阵 GASF（Gramian Angular Summation Field） 为

$$GASF = [\cos(\phi_i + \phi_j)]_{1 \le i,j \le n}$$

于是，

$$GASF = [\cos(\phi_i) \cdot \cos(\phi_j) - \sin(\phi_i) \cdot \sin(\phi_j)]_{n \times n}$$

令 $\tilde{X} = (\cos(\phi_1), \cdots, \cos(\phi_n))^T$ ，可以得到

$$GASF = \tilde{X} \cdot \tilde{X}^T - \sqrt{I - \tilde{X}^2} \cdot \sqrt{I - \tilde{X^T}^2}$$

以上的都是 element 乘法和加法， I 表示单位矩阵。它的对角矩阵是

$$diag(GASF) = \{\cos(2\phi_1), \cdots, \cos(2\phi_n)\}$$

$$= \{2\cos^2(\phi_1) - 1, \cdots, 2\cos^2(\phi_n) - 1)\} = \{GASF_{ii}\}_{1 < i < n}.$$

如果是使用 min-max normalization 的话，是可以从 diag(GASF) 反推出 $\tilde{x}_i$ 的。因为，$2\tilde{x}_i^2 - 1 = 2\cos^2(\phi_i) - 1 = GASF_{ii}$ ，可以得到 $y_i = \sqrt{(GASF_{ii} + 1)/2}$ 。

定义 GADF（Gramian Angular Difference Field）如下：

$$GADF = [\sin(\phi_i - \phi_j)]_{1\leq i,j\leq n}$$

$$= [\sin(\phi_i) \cdot cos(\phi_j) - \cos(\phi_i) \cdot \sin(\phi_j)]_{1\leq i,j\leq n}$$

$$= \sqrt{1 - X^2} \cdot X^T - X \cdot \sqrt{1 - (X^T)^2}.$$

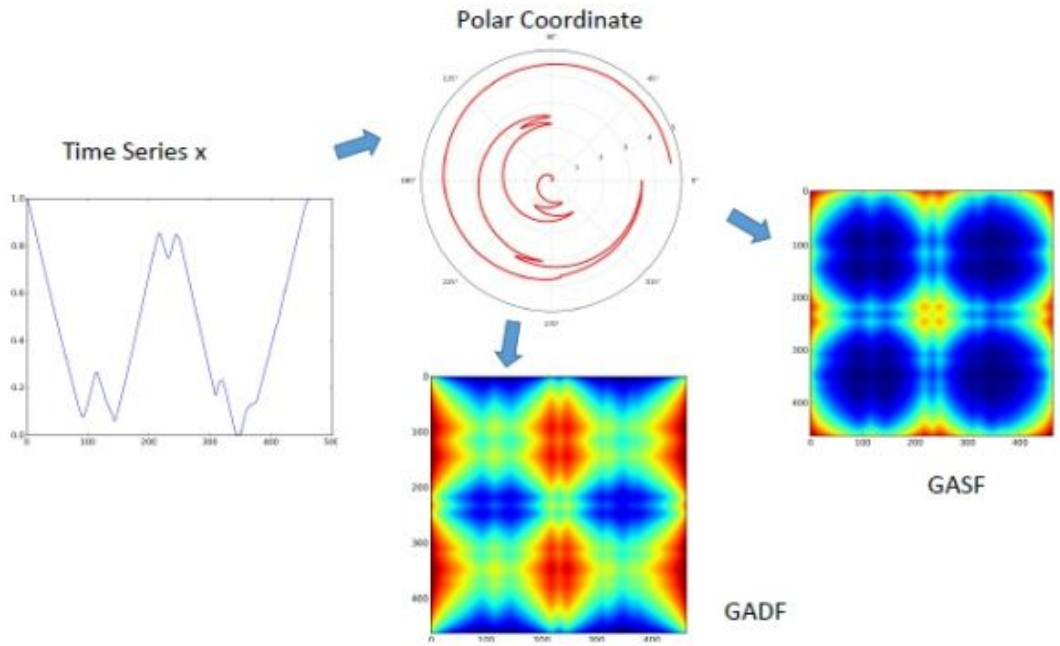


Figure 1: Illustration of the proposed encoding map of Gramian Angular Fields. X is a sequence of rescaled time series in the 'Fish' dataset. We transform X into a polar coordinate system by eq. (3) and finally calculate its GASF/GADF images with eqs. (5) and (7). In this example, we build GAFs without PAA smoothing, so the GAFs both have high resolution.

Markov Transition Field（MTF）

除了 GSAF 和 GSDF 之外，**《Imaging Time Series to Improve Classification and Imputation》，《Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks》，《Encoding Temporal Markov Dynamics in Graph for Time Series Visualization》**也提到了把时间序列转换成矩阵 Image 的算法 MTF。在 pyts 开源工具库里面，也提到了 MTF 算法的源码。

。

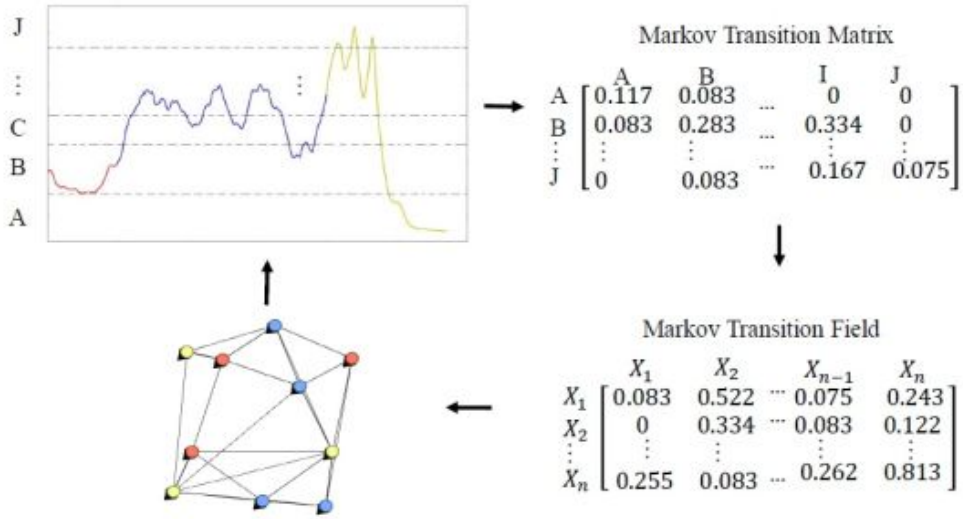


Figure 1: Illustration of the proposed encoding map of a Markov Transition Field. X is a sequence of the typical time series in the 'ECG' dataset. X is first discretized into Q quantile bins. Then we calculate its Markov Transition Matrix W and finally build its MTF M by Equation (2). We reduce the image size from 96×96 to 48×48 by averaging the values in each non-overlapping 2×2 patch.

除此之外，我们也能够计算一个迁移概率矩阵

马尔科夫构图方法：

1.每一个时间节点作为图节点

2.PageRank作为节点权重

时间序列的**降维方法**有两种：

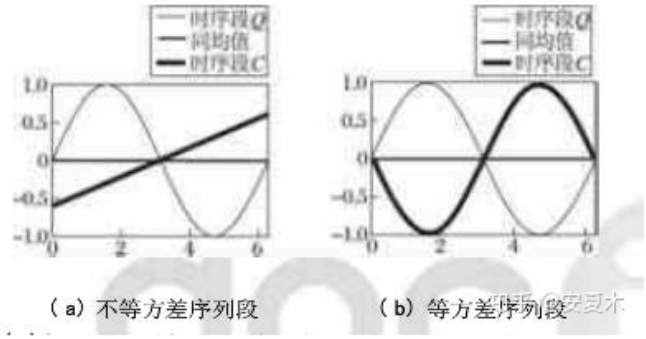
1. **分段聚合（PAA）**：使用局部平均等方法，把时间序列进行降维；

符号聚合近似定义

> 给定长度为 m 的时间序列 $Q = q_1, q_2, \ldots, q_m$ ，把它变成长度为 w 的数据序列 $Q' = q_1', q_2', \ldots, q_w'$ ，其中 $w \leq m$ 。那么，该方式是一个压缩比为 $k = \frac{m}{w}$ 的时间序列数据降维过程，并且 q_i' 满足如下公式：
>
> $$q_i' = \frac{1}{k} \sum_{j=k(i-1)+1}^{k*i} q_j, \quad i = 1, 2, \ldots, w$$

时间序列形态特征

但是会有这样一个问题，对于时间序列，其实如果只用均值来表示，会丢失一部分信息。比如下面两个图形，均值相同，方差也相同的两个序列，但实际形态不同。



（a）不等方差序列段　　（b）等方差序列段

上面两个图形从统计意义来看，其实均值相同，分为了方差相等和不相等两组，从这里可以看到，只用统计值还不能完全将一个时间序列明确的表示出来，还要使用序列的形态特征来表示。形态特征可以使用斜率、角度等来衡量。

根据上述时间序列，对于每一段压缩后的子序列的斜率值公式为：

$$\bar{q_i} = \frac{k \sum_{j=j_0}^{k*i} j q_j - (\sum_{j=j_0}^{k*i} j)(\sum_{j=j_0}^{k*i} q_j)}{k \sum_{j=j_0}^{k*i} j^2 - (\sum_{j=j_0}^{k*i} j)^2}$$

2. **核变换（Kernel）**：使用 Bivariate Gaussian 核或者均值核来把时间序列进行降维。

在把时间序列进行可视化之后，对于时间序列分类的场景，就可以使用 CNN 的技术方案来做了。如下图所示：

- **Blurring** on MTF is fulfilled by averaging the values in each non-overlapping $m \times m$ patch with a blurring kernel, typically a Bivariate Gaussian $\{\frac{1}{2\pi\sigma^2} e^{-\frac{x_1^2+x_2^2}{2\sigma^2}}\}_{m \times m}$ or a simple average kernel $\{\frac{1}{m^2}\}_{m \times m}$. That is, we aggregate the transition probabilities in each subsequence of length m together using a convolution operation with a smaller kernel. The final output size $S = \lceil \frac{n}{m} \rceil$.

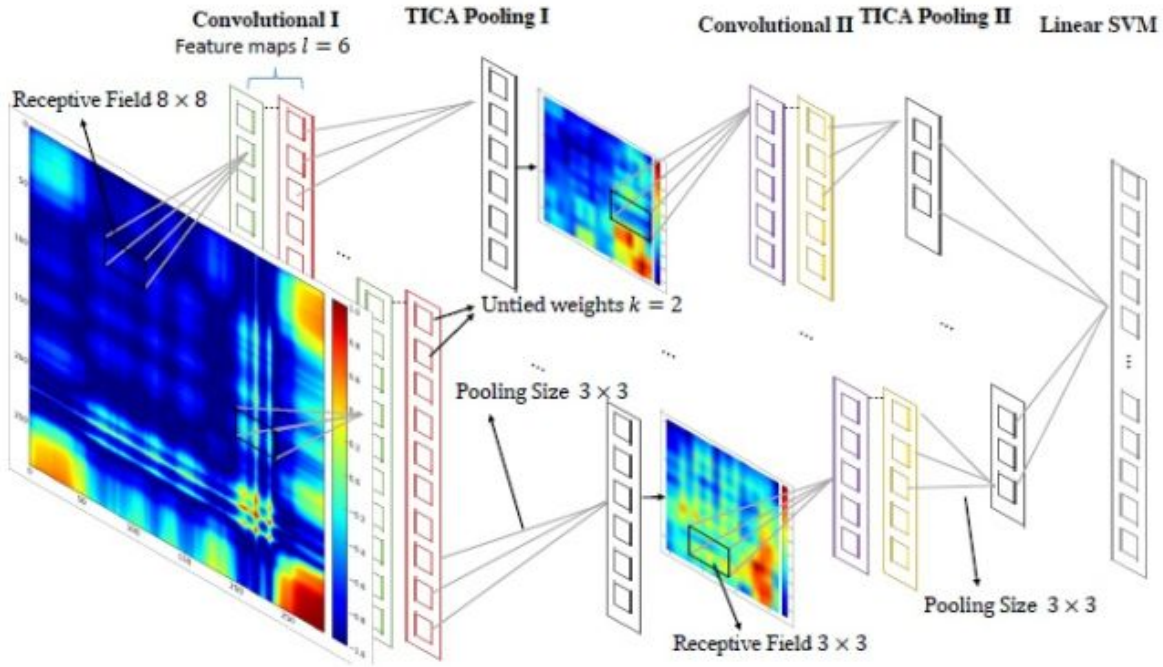


Figure 3: Structure of the tiled convolutional neural network. We fix the size of receptive field to 8×8 in the first convolutional layer and 3×3 in the second convolutional layer. Each TICA pooling layer pools over a block of 3×3 input units in the previous layer without wraparound the boarders to optimize for sparsity of the pooling units. The number of pooling units in each map is exactly the same as the number of input units. The last layer is a linear SVM for classification. We construct this network by stacking two Tiled CNNs, each with 6 maps ($l = 6$) and tiling size $k = 2$.

其实验数据效果如下：

Table 2: Summary statistics of standard dataset and comparative results

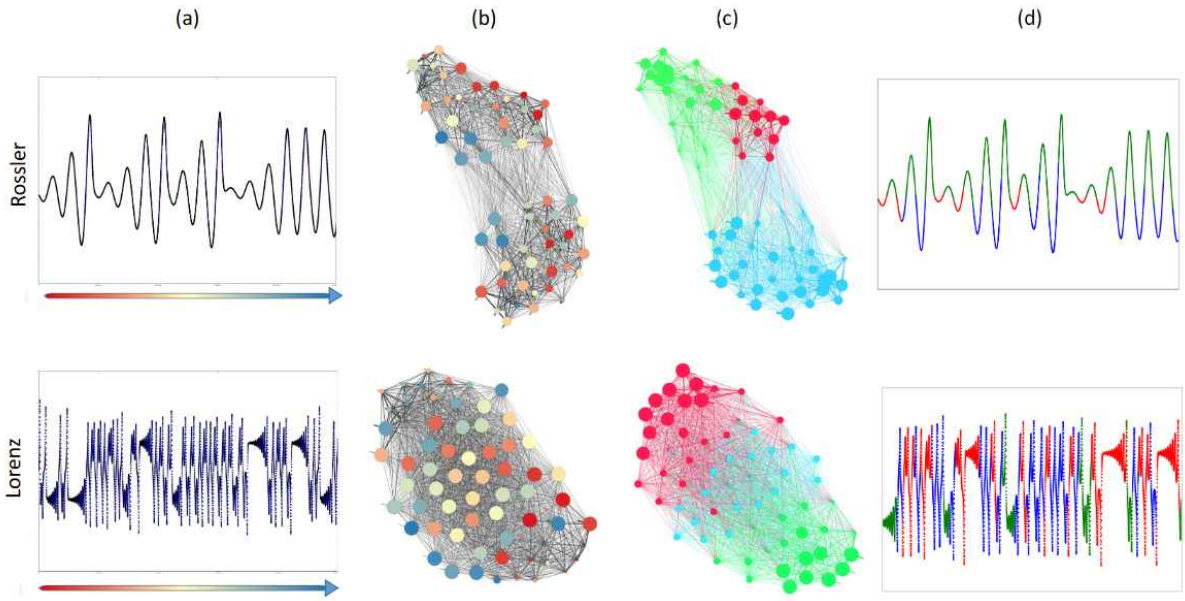
DATASET	CLASS	TRAIN	TEST	LENGTH	1NN-EUCLIDEAN	1NN-DTW	FAST SHAPELET	BOP	SAX-VSM	GAF-MTF
50words	50	450	455	270	0.369	**0.242**	0.4429	0.466	N/A	0.284
Adiac	37	390	391	176	0.389	0.391	0.514	0.432	0.381	**0.307**
Beef	5	30	30	470	0.467	0.467	0.447	0.433	**0.033**	0.3
Coffee	2	28	28	286	0.25	0.18	0.067	0.036	**0**	**0**
ECG200	2	100	100	96	0.12	0.23	0.227	0.14	0.14	**0.08**
FaceAll	14	560	1,690	131	0.286	**0.192**	0.402	0.219	0.207	0.223
Lightning2	2	60	61	637	0.246	**0.131**	0.295	0.164	0.196	0.18
Lightning7	7	70	73	319	0.425	**0.274**	0.403	0.466	0.301	0.397
OliveOil	4	30	30	570	0.133	0.133	0.213	0.133	**0.1**	0.167
OSULeaf	6	200	242	427	0.483	0.409	0.359	0.236	**0.107**	0.446
SwedishLeaf	15	500	625	128	0.213	0.21	0.27	0.196	0.251	**0.293**
Yoga	2	300	3,000	426	0.17	0.164	0.249	0.17	0.164	**0.13**

[2]《Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks》

https://readpaper.com/pdf-annotate/note?pdfId=704838823904317440&noteId=704838849271468032

[3]《Encoding Temporal Markov Dynamics in Graph for Time Series Visualization》

社团检测



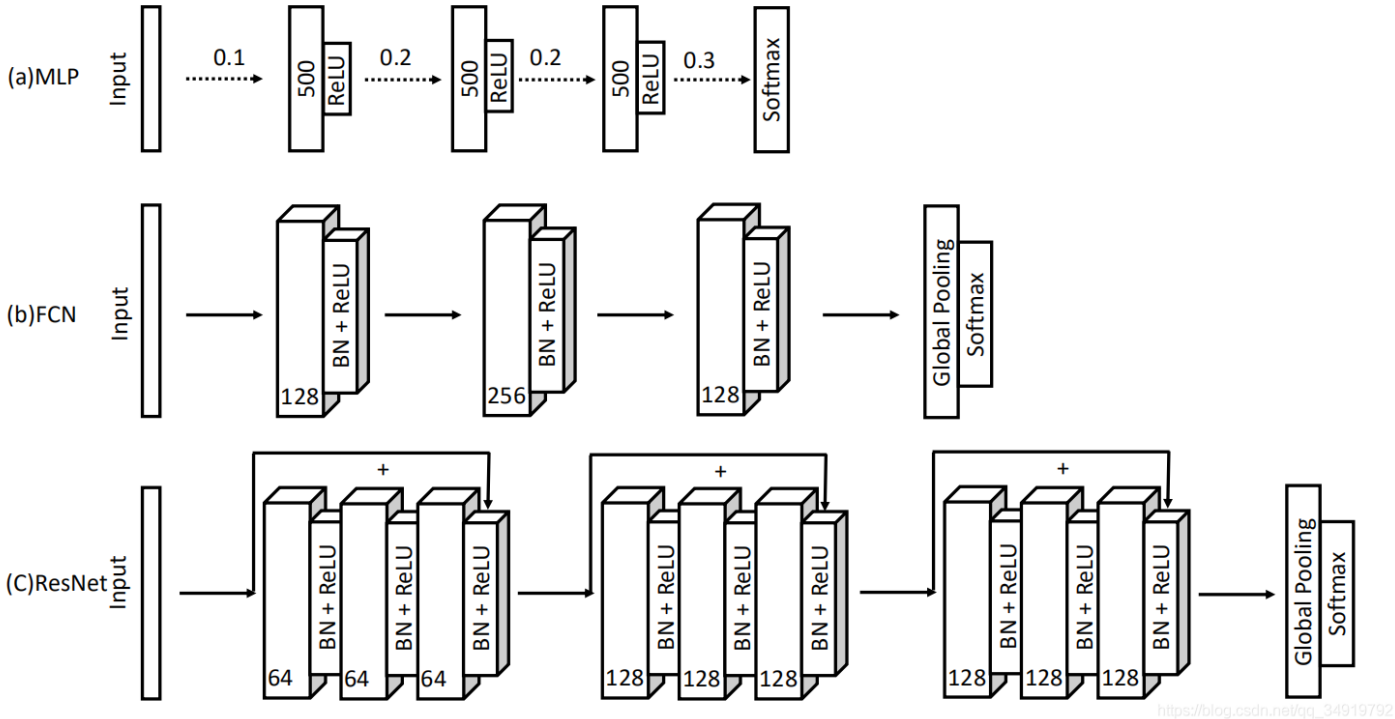
异常检测任务

- Select the most isolated vertices, denote as the set H.
- The top k vertices with the smallest clustering coefficient are selected, denote as the set S

找到最孤立的节点，找到前k个具有最小聚类系数的节点

图分类任务

[4]Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline



一维卷积

[5]Classification of Time-Series Images Using Deep Convolutional Neural Networks

真正使用了深度卷积

and Local Binary Patterns (LBP) on RP to extract texture features from time series, and then used SVM classifiers.[25,26] The difference between the latter and our work is that they used a traditional classification framework with hand-crafted features and separated feature extraction and classification steps. While our proposed CNN-based framework, automatically learns the texture features that are useful for the classification layer. The joint learning of feature representation and classifier offered by CNN increases the classification performance.

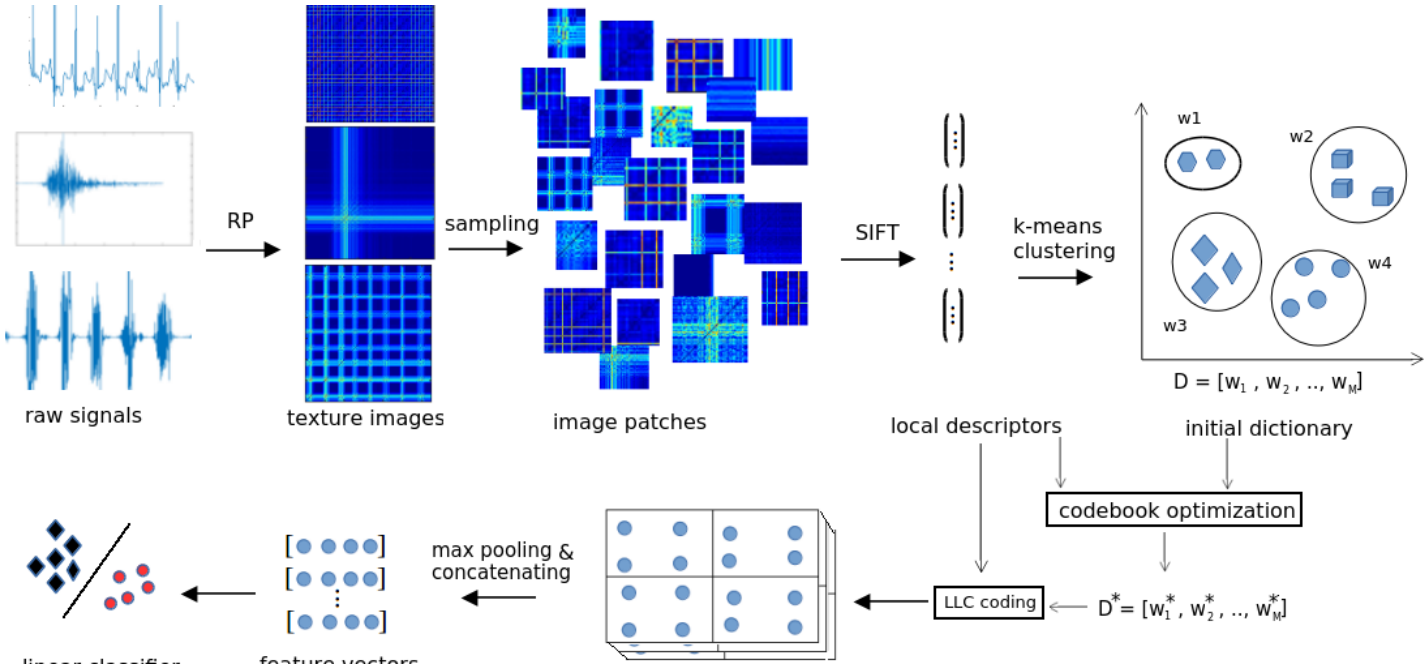

RP
sampling
SIFT
k-means clustering
w1
w2
w3
w4
D = [w1 , w2 , .., wM]
raw signals
texture images
image patches
local descriptors
initial dictionary
codebook optimization
LLC coding
D*= [w1*, w2*, .., wM*]
max pooling & concatenating
feature vectors

TABLE 1. Performance comparison of proposed models with the rest.

Dataset	Existing SOTA	LSTM-FCN	Refined LSTM-FCN	ALSTM-FCN	Refined ALSTM-FCN
Adiac	0.8570 [11]	0.8593	0.8849	0.8670	0.8900*
ArrowHead	0.8800 [11]	0.9086	0.9029	0.9257*	0.9200
Beef	0.9000 [38]	0.9000	0.9330	0.9333*	0.9333*
BeetleFly	0.9500 [11, 19]	0.9500	1.0000*	1.0000*	1.0000*
BirdChicken	0.9500 [11, 17]	1.0000*	1.0000*	1.0000*	1.0000*
Car	0.9330 [11]	0.9500	0.9670	0.9667	0.9833*
CBF	1.0000 [11]	0.9978	1.0000*	0.9967	0.9967
ChloConc	0.8720 [11]	0.8099	1.0000*	0.8070	0.8070
CinC_ECG	0.9949 [8]	0.8862	0.9094	0.9058	0.9058
Coffee	1.0000 [19, 39]	1.0000*	1.0000*	1.0000*	1.0000*
Computers	0.8480 [11]	0.8600	0.8600	0.8640*	0.8640*
Cricket_X	0.8210 [11]	0.8077	0.8256*	0.8051	0.8051
Cricket_Y	0.8256 [8]	0.8179	0.8256*	0.8205	0.8205
Cricket_Z	0.8154 [8]	0.8103	0.8257	0.8308	0.8333*
DiaSizeRed	0.9670 [20]	0.9673	0.9771*	0.9739	0.9739
DistPhxAgeGp	0.8350 [11]	0.8600	0.8600	0.8625*	0.8600
DistPhxCorr	0.8200 [11]	0.8250	0.8217	0.8417*	0.8383
DistPhxTW	0.7900 [11]	0.8175	0.8100	0.8175	0.8200*
Earthquakes	0.8010 [11]	0.8354*	0.8261	0.8292	0.8292
ECG200	0.9200 [11]	0.9000	0.9200*	0.9100	0.9200
ECG5000	0.9482 [19]	0.9473	0.9478	0.9484	0.9496*
ECGFiveDays	1.0000 [19, 39]	0.9919	0.9942	0.9954	0.9954
ElectricDevices	0.7993 [17]	0.7681	0.7633	0.7672	0.7672
FaceAll	0.9290 [11]	0.9402	0.9680	0.9657	0.9728*
FaceFour	1.0000 [16, 19]	0.9432	0.9772	0.9432	0.9432
FacesUCR	0.9580 [11]	0.9293	0.9898*	0.9434	0.9434
FiftyWords	0.8198 [20]	0.8044	0.8066	0.8242	0.8286*
Fish	0.9890 [11]	0.9829	0.9886	0.9771	0.9771
FordA	0.9727 [19]	0.9272	0.9733*	0.9267	0.9267
FordB	0.9173 [39]	0.9180	0.9186*	0.9158	0.9158
Gun_Point	1.0000 [19, 38]	1.0000*	1.0000*	1.0000*	1.0000*
Ham	0.7810 [11]	0.7714	0.8000	0.8381*	0.8000
HandOutlines	0.9487 [19]	0.8930	0.8870	0.9030	0.9030
Haptics	0.5510 [11]	0.5747*	0.5584	0.5649	0.5584
Herring	0.7030 [11]	0.7656*	0.7188	0.7500	0.7656*
InlineSkate	0.6127 [19]	0.4655	0.5000	0.4927	0.4927
InsWngSnd	0.6525 [8]	0.6616	0.6696	0.6823*	0.6818
ItPwDmd	0.9700 [11]	0.9631	0.9699	0.9602	0.9708*
LrgKitApp	0.8960 [11]	0.9200*	0.9200*	0.9067	0.9120
Lighting2	0.8853 [20]	0.8033	0.8197	0.7869	0.7869
Lighting7	0.8630 [11]	0.8356	0.9178*	0.8219	0.9178*
Mallat	0.9800 [11]	0.9808	0.9834	0.9838	0.9842*
Meat	1.0000 [11]	0.9167	1.0000*	0.9833	1.0000*
MedicalImages	0.7920 [11]	0.8013	0.8066*	0.7961	0.7961
MidPhxAgeGp	0.8144 [16]	0.8125	0.8150	0.8175*	0.8075
MidPhxCorr	0.8076 [16]	0.8217	0.8333	0.8400	0.8433*
MidPhxTW	0.6120 [11]	0.6165	0.6466	0.6466*	0.6316
MoteStrain	0.9500 [11]	0.9393	0.9569*	0.9361	0.9361
NonInv_Thor1	0.9610 [11]	0.9654	0.9657	0.9751	0.9756*
NonInv_Thor2	0.9550 [11]	0.9623	0.9613	0.9664	0.9674*
OliveOil	0.9333 [19]	0.8667	0.9333	0.9333	0.9667*
OSULeaf	0.9880 [11]	0.9959*	0.9959*	0.9959*	0.9917

OSULeaf	0.9880 [11]	0.9959*	0.9959*	0.9959*	0.9917
PhalCorr	0.8300 [11]	0.8368	0.8392*	0.8380	0.8357
Phoneme	0.3492 [8]	0.3776*	0.3602	0.3671	0.3623
Plane	1.0000 [19, 20]	1.0000*	1.0000*	1.0000*	1.0000*
ProxPhxAgeGp	0.8832 [38]	0.8927*	0.8878	0.8878	0.8927*
ProxPhxCorr	0.9180 [11]	0.9450*	0.9313	0.9313	0.9381
ProxPhxTW	0.8150 [17]	0.8350	0.8275	0.8375*	0.8375*
RefDev	0.5813 [38]	0.5813	0.5947*	0.5840	0.5840
ScreenType	0.7070 [11]	0.6693	0.7073	0.6907	0.6907
ShapeletSim	1.0000 [17, 19]	0.9722	1.0000*	0.9833	0.9833
ShapesAll	0.9183 [19]	0.9017	0.9150	0.9183	0.9217*
SmlKitApp	0.8030 [11]	0.8080	0.8133*	0.7947	0.8133*
SonyAIBOI	0.9850 [11]	0.9817	0.9967	0.9700	0.9983*
SonyAIBOII	0.9620 [11]	0.9780	0.9822*	0.9748	0.9790
StarlightCurves	0.9796 [8]	0.9756	0.9763	0.9767	0.9767
Strawberry	0.9760 [17]	0.9838	0.9864	0.9838	0.9865*
SwedishLeaf	0.9664 [19]	0.9792	0.9840	0.9856*	0.9856*
Symbols	0.9668 [17]	0.9839	0.9849	0.9869	0.9889*
Synth_Cntr	1.0000 [8, 20]	0.9933	1.0000*	0.9900	0.9900
ToeSeg1	0.9737 [8]	0.9825	0.9912*	0.9868	0.9868
ToeSeg2	0.9615 [17]	0.9308	0.9462	0.9308	0.9308
Trace	1.0000 [19, 20]	1.0000*	1.0000*	1.0000*	1.0000*
Two_Patterns	1.0000 [11, 20]	0.9968	0.9973	0.9968	0.9968
TwoLeadECG	1.0000 [8, 20]	0.9991	1.0000*	0.9991	1.0000*
uWavGest_X	0.8308 [16]	0.8490	0.8498	0.8481	0.8504*
uWavGest_Y	0.7585 [8]	0.7672*	0.7661	0.7658	0.7644
uWavGest_Z	0.7725 [16]	0.7973	0.7993	0.7982	0.8007*
uWavGestAll	0.9685 [20]	0.9618	0.9609	0.9626	0.9626
Wafer	1.0000 [19, 38]	0.9992	1.0000*	0.9981	0.9981
Wine	0.8890 [11]	0.8704	0.8890	0.9074*	0.9074*
WordsSynonyms	0.7790 [20]	0.6708	0.6991	0.6677	0.6677
Worms	0.8052 [19]	0.6685	0.6851	0.6575	0.6575
WormsTwoClass	0.8312 [38]	0.7956	0.8066	0.8011	0.8011
yoga	0.9183 [17]	0.9177	0.9163	0.9190	0.9237*
Count	-	43	65	51	57
MPCE	-	0.0318	0.0283	0.0301	0.0294
Arith. Mean	-	-	2.1529	-	2.5647
Geom. Mean	-	-	1.8046	-	1.8506