Increasing dimensionality of time series data using Gramian Angular Fields and Markov Transition Fields to perform classification using CNN's

Deepak Ravikumar

Purdue University

Table of Contents

Abstract

This project uses two mathematical techniques, namely Gramian Angular Fields (GAF) and Markov Transition Fields (MTF), to convert 1D time series data to a 2D image more suited for classification using convolutional neural nets. The technique is applied on various time series data sets from [1] and compares the results obtained with results from [2]. The code for this project was run on GPUs using Google colaboratory [3] and the speedups was tabulated. The project also investigates the application of this technique on stock market data to identify profitable stocks.

Keywords: Gramian Angular Field, Markov Transition Field, CNN, 1D to 2D

Encoding Time Series to Images

In this section we briefly describe the two mathematical methods used in the project to convert time series data to images, namely, Gramian Angular field (GAF), in which we represent time series in a polar coordinate system instead of the typical Cartesian coordinates and Markov Transition Field (MTF)

Gramian Angular Field (GAF)

This section, in order to keep things simple will discuss how to calculate a GAF. Firstly we need to normalize the data.

$$\tilde{x}_i = \frac{((x_i - max(X)) + (x_i - min(X))}{\max(X) - \min(X)} \tag{1.1}$$

Where X is a time series denoted by $X = \{x_1, x_2, .., x_n\}$. This is represented in the polar coordinates using the equations below equation (1.2)

$$\begin{cases} \phi = \arccos(\tilde{x}_i), -1 \leq \tilde{x}_i \leq 1, \tilde{x}_i \in X \\ r = \dfrac{t_i}{N}, t_i \in \mathbb{N} \end{cases} \tag{1.2}$$

Using the rescaled time series and having converted it into polar coordinates, we can exploit the perspective by considering the trigonometric sum between each point. The GAF is defined below

$$G = \begin{bmatrix} \cos(\phi_1 + \phi_1) & \cdots & \cos(\phi_1 + \phi_n) \\ \cos(\phi_1 + \phi_2) & \cdots & \cos(\phi_2 + \phi_n) \\ \vdots & \ddots & \vdots \\ \cos(\phi_1 + \phi_n) & \cdots & \cos(\phi_n + \phi_n) \end{bmatrix} \tag{1.3}$$

$$= X' \cdot X - \sqrt{1 - X^2}' \cdot \sqrt{1 - X^2} \tag{1.4}$$

Markov Transition Field (MTF)

Given a time series X, we identify its Q quantile bins and assign each xi to the corresponding bins q_j (j belongs to the interval $[1,Q]$). Thus we construct a $Q \times Q$ weighted adjacency matrix W by counting transitions among quantile bins in the manner of a first-order Markov chain along the time axis. $w_{i,j}$ is given by the frequency with which a point in the quantile q_j is followed by a point in the quantile q_i. After normalization by $\Sigma w_{i,j} = 1$, W is the Markov transition matrix given by the equation (1.5).

$$M = \begin{bmatrix} w_{ij|x_1 \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_1 \in q_i, x_n \in q_j} \\ w_{ij|x_2 \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_2 \in q_i, x_n \in q_j} \\ \vdots & \ddots & \vdots \\ w_{ij|x_n \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_n \in q_i, x_n \in q_j} \end{bmatrix} \tag{1.5}$$

Network Architecture and System Description

This section describes the system architecture used in the project. The GAF and MTF transformations were performed before the data is fed into the network. The time series data is converted into an image using GAF and MTF. Each of GAF and MTF generates one image.

The input data is fed into the GAF and MTF computation, from which we obtain two images. One from MTF and one from GAF. These are combined to create a two channel input image. Two image sizes, 96 x 96 and 128 x 128 were used. The size of the input image to the CNN depended on the input data's feature size.

The generated image of two channels was fed into a CNN. The CNN consists of 3 convolution layers followed by 2 dense layers.

1. **Layer 1:** The first convolution layer has a receptive field of size 8x8 and has 32 kernels/filters. This is followed by a Relu activation. This is followed by a max pooling layer of size 3x3. A dropout of 50% is used in this layer.
2. **Layer 2:** This layer's convolution has 64 kernels/filters of size 3x3. Which is followed by a Relu activation layer, followed by a 3x3 max pooling layer and uses 50% dropout.
3. **Layer 3:** This layer's convolution has 128 kernels/filters of size 3x3. Which is followed by a Relu activation layer, followed by a 3x3 max pooling layer and uses 50% dropout.
4. **Layer 4:** Layer 4 flattens the output of layer 3 and uses a 512 dense network with a Relu activation and 50% dropout
5. **Layer 5:** This is the output layer which is fully dense and the activation is softmax.

This network was used to classify various timeseries datasets, its accuracy and results are discussed in the ***Results*** section of this report.

The same network but with layers 2 and 3 with dropouts removed was used on stock market data. The results of which are discussed in the ***Results*** section of this report. Before we look into the results, I want to briefly go over the way the stock data was prepared before presenting it to this network.

The result we want to achieve is to predict the price of the stock, but this is useful however a more useful output would be weather if the price is going to go up or down, and this is what the project does. So by reframing the question we have taken a regression problem and converted that into a classification problem which suits the network and the system designed above.

To run the stock market data (nasdaq100) on the system, it was preprocessed in the following way. Let $X = \{ x_1, x_2, x_3, x_4 \ldots x_n\}$ represent the n timeseries data samples of the stock market data, which is rearranged into a matrix

$$D = \begin{pmatrix} x_{1,1} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots \\ x_{97,1} & \cdots & x_{97,m} \end{pmatrix} \tag{1.6}$$

Where m is

$$m = \left\lfloor \frac{n}{97} \right\rfloor \tag{1.7}$$

Each column in matrix D represents the one 'cluster' of stock data, and we have m such samples. As we will see later we will train on each of these m samples. The reason 97 was chosen was because it is 96 (image size for the network) + 1 (used to set the label for the sample).

The next step was to assign the label to the sample. To do that we create a vector Y such that

$$Y = \{y_1, y_2, y_3, ..., y_m\} \tag{1.8}$$

$$y_i = \begin{cases} 1 & \text{if } x_{97,i} - x_{96,i} >= 0 \text{ for } 1 \le i \le m \\ 0 & \text{otherwise} \end{cases} \tag{1.9}$$

Now that we have the labels and samples we use $\{D, Y\}$ as our labeled data set to train and test the network. This generates 418 samples each having 97 features. Out of the 418 samples the first 313 samples are chosen as training samples (75% training set). The rest are chosen as the testing samples.

Dataset details

This project uses data sets available from [1], the table below describes the details of the data set, namely, number of classes, testing set size, training set size and length.

Table 1: Dataset details

Database	Classes	Training Set	Testing Set	Length
Adiac	37	390	391	176
Beef	5	30	30	470
Coffee	2	28	28	286

Database	Classes	Training Set	Testing Set	Length
ECG200	2	100	100	96
FaceAll	14	560	1690	131
FiftyWords	50	450	455	270
Lightning2	2	60	61	637
Lightning7	7	70	73	319
OliveOil	4	30	30	570
OSULeaf	6	200	242	427
SwedishLeaf	15	500	625	128
Yoga	2	300	3000	426

Results

Part 1: Comparison between reference paper and the project

This section provides a detailed quantitative results of the application of the above described technique to classify time series data sets on 12 different datasets from [1]. Table 2 shows the result of applying the system described in the previous sections on various data sets, Training accuracy (column 'Training acc.') is the accuracy in classification on the training set and the corresponding error in classification is shown in the column 'Train error'. Similarly testing accuracy is the classification accuracy on the test set, and test error shows the corresponding error.

Table 3 and Figure 1 show the comparison of error rates between the paper [2] and the results from the project, looking at the graph we can see that the system described in the project has identical or similar error rates to the reference.

Table 2: Accuracy of the system of various time series data sets

Dataset	Training acc.	Testing acc.	Train error	Test error
Adiac	0.5872	0.5243	0.4128	0.4757
Beef	0.8333	0.6333	0.1667	0.3667
Coffee	0.9643	1.0000	0.0357	0.0000
ECG200	1.0000	1.0000	0.0000	0.0000
FaceAll	0.9250	0.7260	0.0750	0.2740
FiftyWords	0.8911	0.7297	0.1089	0.2703
Lightning2	1.0000	1.0000	0.0000	0.0000
Lightning7	0.8714	0.6027	0.1286	0.3973
OliveOil	0.4333	0.4000	0.5667	0.6000
OSULeaf	1.0000	0.6074	0.0000	0.3926
SwedishLeaf	0.8820	0.8512	0.1180	0.1488
Yoga	0.7667	0.7437	0.2333	0.2563

Table 3: Comparing the error rates between the system and the one descried in [2]

Dataset	Error Results observed	Error Results from [2]
Adiac	0.4757	0.307
Beef	0.3667	0.3
Coffee	0.0000	0
ECG200	0.0000	0.08
FaceAll	0.2740	0.223
FiftyWords	0.2703	0.284
Lightning2	0.0000	0.18
Lightning7	0.3973	0.397
OliveOil	0.6000	0.167
OSULeaf	0.3926	0.446
SwedishLeaf	0.1488	0.093
Yoga	0.2563	0.16

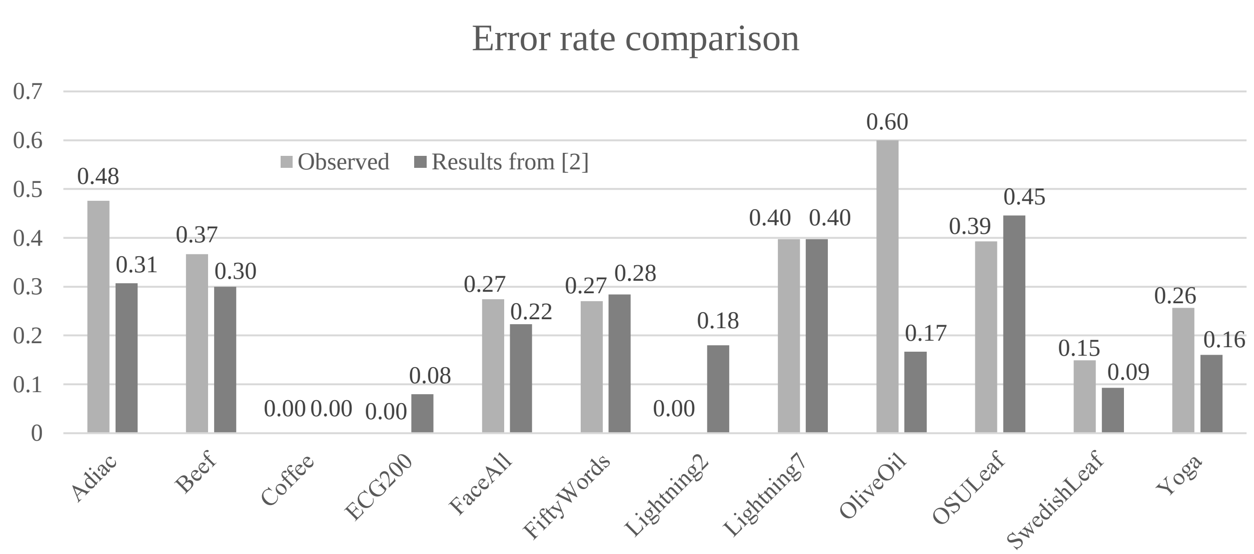

Figure 1: Graph visualizing and comparing the error rates between the system and the one descried in [2]

Part 2: Comparison with 1D time series data

The section above discussed the results obtained when the time series data was converted to a 2D image using GAF and MTF. To compare the effectiveness of the technique, a neural net was trained in the 1D data. This new network will be referred to as one-dim net. To keep the comparison fair, the number of parameters in each stage of the CNN and the one-dim net were kept the same. The number of hidden layers were also kept the same. The only difference between the CNN and the one-dim net is that the convolution was removed. The one-dim net's input size was set to the size of the data. The results from running the one-dim net on google colaboratory are on various data sets are shown in Table 4 and Figure 2.

Table 4: Results of 'one-dim net' when run on various data sets.

Database	GPU (seconds)	Training acc.	Testing acc.	Train error	Test error	Epochs	Batch Size	Overfitting
Adiac	80.5676	0.6333	0.6061	0.3667	0.3939	500	32	0.0272
Beef	18.9190	0.9000	0.7667	0.1000	0.2333	1000	32	0.1333
Coffee	5.9900	0.9643	0.9643	0.0357	0.0357	250	32	0.0000
ECG200	14.9864	1.0000	1.0000	0.0000	0.0000	250	32	0.0000
FaceAll	55.6137	0.9982	0.8728	0.0018	0.1272	250	32	0.1254
FiftyWords	48.4395	0.9622	0.6923	0.0378	0.3077	250	32	0.2699
Lightning2	11.0542	1.0000	1.0000	0.0000	0.0000	500	32	0.0000
Lightning7	13.8308	1.0000	0.5753	0.0000	0.4247	250	32	0.4247
OliveOil	7.3723	0.3333	0.4000	0.6667	0.6000	250	32	0.0667
OSULeaf	100.7581	1.0000	0.5537	0.0000	0.4463	1000	32	0.4463
SwedishLeaf	50.9253	0.9720	0.8736	0.0280	0.1264	250	32	0.0984
Yoga	36.6654	0.9500	0.8227	0.0500	0.1773	250	32	0.1273

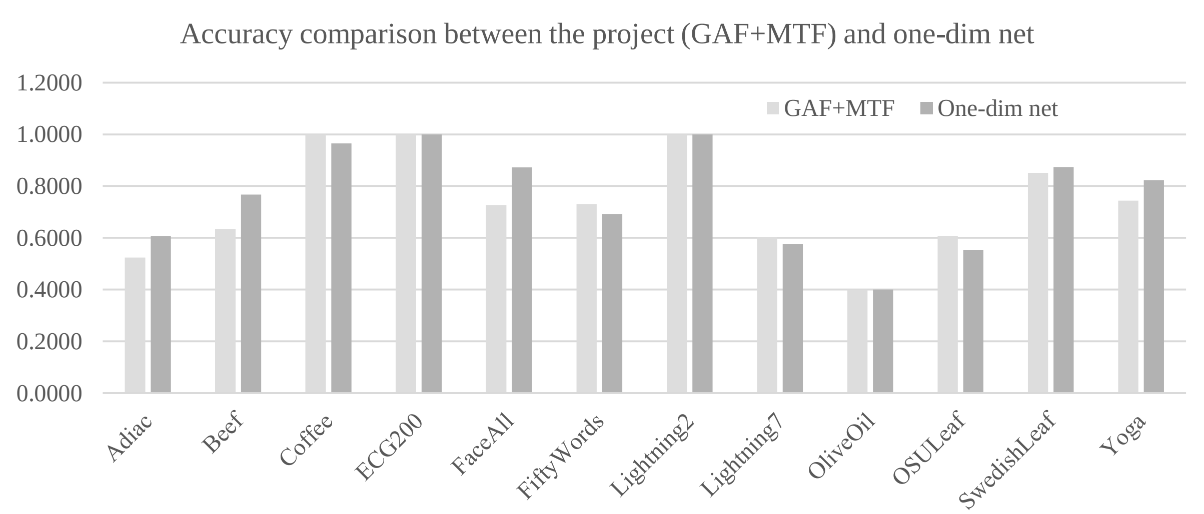

Figure 2: Accuracy comparison between the project (GAF+MTF) and one-dim net

Part 3: Speedup results, when run on Google colaboratory

The code for the project was run on Google's no acceleration runtime and the same code was run on the colaboratory's GPUs and the following speedups were observed see Table 5.

Table 5: Runtime and Speedup observed when run on Google colaboratory

Database	GPU (seconds)	CPU (seconds)	Training acc.	Testing acc.	Epochs	Batch Size	Image Size	Speedup
Adiac	235.7983	6807.7860	0.5872	0.5243	500	32	128	28.8712
Beef	41.9577	1092.8183	0.8333	0.6333	1000	32	128	26.0457
Coffee	12.3368	258.1204	0.9643	1.0000	250	32	128	20.9229
ECG200	34.7675	537.7359	1.0000	1.0000	250	32	96	15.4666
FaceAll	214.1824	4799.5617	0.9250	0.7260	250	32	128	22.4088
FiftyWords	144.2903	3993.7420	0.8911	0.7297	250	32	128	27.6785
Lightning2	27.0638	1072.5493	1.0000	1.0000	500	32	128	39.6304
Lightning7	49.1692	625.9870	0.8714	0.6027	250	32	128	12.7313
OliveOil	41.9178	271.1055	0.4333	0.4000	250	32	128	6.4676
OSULeaf	246.8397	6558.4963	1.0000	0.6074	1000	32	128	26.5699
SwedishLeaf	161.3626	4401.6564	0.8820	0.8512	250	32	128	27.2780
Yoga	229.3734	2543.9712	0.7667	0.7437	250	32	128	11.0910

Part 4: Results of applying the system on stock market data

The same system with layers 2 and 3's dropouts removed was used on stock market data to predict if the stock price rises or falls. The data was organized into 97 elements chunks as described in "Network Architecture and System Description", which acts as a memory element needed to predict the stock market data. Each stock was trained separately over the samples so that the features of one stock's performance does not affect the other. The result of this is tabulated in Table 6. The *over-fitting* parameter is the difference between training accuracy and testing accuracy.

Table 6: Performance of system on stock market data

Stock	GPU Runtime (seconds)	Training acc.	Testing acc.	Train error	Test error	Epochs	Batch Size	Over-fitting (0-1)	Image Size
AAL	14.6050	0.6230	0.6000	0.3770	0.4000	50	32	0.0230	96
AAPL	52.2676	0.9617	0.5619	0.0383	0.4381	200	32	0.3998	96
AMZN	17.0139	0.6454	0.4952	0.3546	0.5048	50	32	0.1502	96

Stock	GPU Runtime (seconds)	Training acc.	Testing acc.	Train error	Test error	Epochs	Batch Size	Over-fitting (0-1)	Image Size
EA	17.1242	0.6070	0.4667	0.3930	0.5333	50	32	0.1403	96
SIRI	17.7047	0.7732	0.7524	0.2268	0.2476	50	32	0.0208	96
TSLA	17.7683	0.6645	0.5238	0.3355	0.4762	50	32	0.1407	96

Conclusion

The project used GAF and MTF first proposed for use in time series classification by [2] and also extended the used case for stock market data. The system performs as expected when applied to standard time series data sets available from [1], and as seen from ***Part 1: Comparison between reference paper and the project*** the system's performance is very similar to [2]. The result of using GAF+MTF and using the **2D reconstruction did not perform any better** see Figure 2. The one-dim net performed at least as good or better than the GAF+MTF on 8/12 (75%) datasets tested on.

The second conclusion that needs to be drawn is the acceleration on Google colaboratory, using Google's GPU runtime environment for the system described provided on average 22.0968x acceleration and a highest of 39.6304x acceleration on the Lightning2 dataset when compared to running on a CPU. The final conclusion is from applying the system on stock market data, the system performance is slightly worse than when applied to standard time series datasets, LSTM's in general perform better than the method used for the system, however a combination would work better.

Source Code

Code for the project is available at *https://github.com/DeepakTatachar/629-Project*

To run the code the following dependencies need to be installed

- *keras*
- *numpy*
- *tensorflow*
- *scikit-image*
- *pyts*
- *scipy*

The code has two major classes, the timeseries class and the reader class.

The timeseries class holds the data, testing images, training images, number of classes, time series converted to GAF and MTF values. It has one major function, to read the input data set and convert it into the right format for training and testing. This is achieved through three functions of the class

- readdataset
- convert_to_GASF
- convert_to_MTF

The other important object is reader, this is a class that reads the dataset based on the type and returns the read dataset and labels, takes the input whether the dataset to be read is a part of the testing or training set. This is done through JSON file, which is parsed to figure out the details of the dataset for example, the type of the dataset, number of classes and so on, the json file is ‘dict.json’.

To run 2-D converted images run main.py, to run the one-dim net run oneDim.py. To run on different datasets change the value of the parameter to time_series.readdataset() function, the value of the argument is the name of the dataset, to find the supported datasets look at the ‘dict.json’ file. The names of the datasets are the same as those used in the Tables shown above see Table 5.

References

[1] A. Bagnall, J. Lines, A. Bostrom, J. Large and E. Keogh, "The Great Time Series Classification Bake Off: a Review and Experimental Evaluation of Recent Algorithmic Advances," *Data Mining and Knowledge Discovery,* vol. 31, no. 3, pp. 606-660, 2017.

[2] Z. Wang and T. Oates, "Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks," *AAAI Workshops,* 2015.

[3] Google, "Google Colaboratory," Google, 2018. [Online]. Available: https://colab.research.google.com.