Enhancing Time-Series Prediction with Temporal Context Modeling: A Bayesian and Deep Learning Synergy

Habib Irani, Vangelis Metsis
Computer Science Department, Texas State University
San Marcos, TX 78666, USA
habibirani@txstate.edu, vmetsis@txstate.edu

Abstract

In time-series classification, conventional deep learning methods often treat continuous signals as discrete windows, each analyzed independently without considering the contextual information from adjacent windows. This study introduces a novel, lightweight Bayesian meta-classification approach designed to enhance prediction accuracy by integrating contextual label information from neighboring windows. Alongside training a deep learning model, we construct a Conditional Probability Table (CPT) during training to capture label transitions. During inference, these CPTs are utilized to adjust the predicted class probabilities of each window, taking into account the predictions of preceding windows. Our experimental analysis, focused on Human Activity Recognition (HAR) time series datasets, demonstrates that this approach not only surpasses the baseline performance of standalone deep learning models but also outperforms contemporary state-of-the-art methods that integrate temporal context into time series prediction.

***Keywords*—**Time Series Classification, Deep Learning, Bayesian Methods, Temporal Context

1 Introduction

Time-series prediction encompasses a spectrum of machine learning tasks, with applications spanning diverse fields such as healthcare, finance, and speech recognition. Within this domain, time-series classification focuses on predicting labels associated with specific segments of time-series data, while time-series forecasting aims to predict future values within the series itself. Our work concentrates on the challenges and advancements within the context of time-series classification.

Traditional machine learning methods, including logistic regression and support vector machines, have demonstrated their utility in time-series classification tasks (de Mattos Neto et al. 2020; Abanda, Mori, and Lozano 2019; Bagnall et al. 2017; Botsch 2023; Kirichenko, Radivilova, and Bulakh 2018). However, these approaches often struggle to explicitly model the intricate temporal dependencies and contextual nuances inherent in time-series data. The rise of deep learning algorithms, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformers, has ushered in a new era of powerful tools for time-series classification (Ismail Fawaz et al. 2019; Alzubaidi et al. 2021). These deep learning architectures excel at recognizing temporal patterns from raw data, leading to significant advancements in the field.

The prevalent approach for time series classifications involves segmenting the continuous data stream into windows of fixed or variable length. Each window is assigned one or more labels, and the model learns the relationship between temporal patterns within the window and the corresponding label. Figure 1 illustrates this process. Deep learning models effectively capture temporal patterns within each window, but current architectures often overlook the broader context beyond the current window. This limitation can hinder performance, particularly when long-term dependencies and trends play a crucial role in accurate classification.

Researchers have explored hybrid deep learning approaches to address this challenge, aiming to capture both short and long-term patterns. LSTNet (Lai et al. 2018), for instance, employs a two-layer architecture with a CNN-based backbone for modeling individual windows and an LSTM with temporal attention to model behavior across multiple windows. However, modeling longer time horizons with increasingly complex temporal patterns often necessitates larger models and extensive training data, raising concerns about computational efficiency and resource requirements.

Our research builds upon the foundations laid by previous contributions, including (Wang, Yan, and Oates 2017; Khan et al. 2021; Zhang, Zhu, and Zhang 2020), who have made significant strides in leveraging deep learning for intricate temporal pattern recognition and representation learning. We also acknowledge the contributions of (Lai et al. 2018) in addressing the challenge of modeling long-term dependencies. However, we recognize the need for methods that balance effectiveness with efficiency and interpretability.

In contrast to complex hybrid deep learning models, we propose a lightweight Bayesian approach for long-term temporal context modeling[1]. Instead of directly analyzing data

[1]https://github.com/imics-lab/TemporalContext-BayesDL

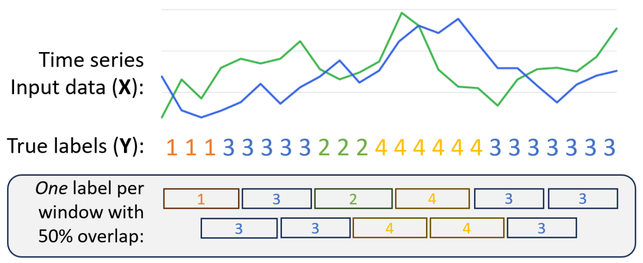
Time series Input data (X):
True labels (Y): 1 1 1 3 3 3 3 3 2 2 2 4 4 4 4 4 4 3 3 3 3 3 3 3
One label per window with 50% overlap:
1
3
2
4
3
3
3
3
4
4
3

Figure 1: For training and inference, time series are segmented into (potentially overlapping) windows, and a label is assigned to each window.

from past windows, we focus on the labels of those windows. By modeling transition probabilities between labels of neighboring windows and integrating this information with deep learning-based predictions for each window, we aim to improve prediction accuracy while maintaining efficiency. We employ Conditional Probability Tables (CPTs) to represent the conditional probabilities of transitioning between different class labels for each window. At inference time, considering the sequence of previous labels through CPTs provides a nuanced understanding of inter-window dependencies, refining initial deep learning-based classifications.

This Bayesian integration enhances adaptability to the dynamic nature of time-series datasets, resulting in improved accuracy and more robust classification results. For example, in human activity recognition (HAR), if consecutive windows are labeled as "running," the subsequent window is more likely to share the same label than transition to "walking downstairs." Our Bayesian approach mitigates the impact of potential misclassifications arising from data artifacts by incorporating contextual information.

In the following sections, we delve deeper into the rationale and algorithmic formulation of our proposed method. We then compare its performance against contemporary deep learning-based time series classification architectures, including LSTNet, to demonstrate its effectiveness and efficiency.

2 Methodology

This section introduces our proposed approach, commencing with a definition of the problem to guide the subsequent methodological steps. We explain the data processing steps, starting from the initial time series segmentation into windows to building the CPTs at training time and then using them to refine the label predictions at inference time.

2.1 Problem Definition

The task at hand revolves around augmenting the accuracy of classification through the adept utilization of temporal dependencies. The dataset under consideration, denoted as $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_M\}$, comprises sequential observations wherein each data window, represented by $\mathbf{X}_i$ (2D matrix of dimensions $(timesteps \times channels)$), is associated with a corresponding class label $\mathbf{y}_i$, represented here as a one-hot-encoded vector. Our primary objective is to employ a Bayesian framework alongside Conditional Probability Tables (CPTs) to effectively model the conditional probabilities of class transitions. Specifically, we seek to ascertain the likelihood of observing a particular class label $\mathbf{y}_i$ given the sequence of preceding labels, denoted as $\mathbf{y}_{i-1}, \mathbf{y}_{i-2}, \ldots, \mathbf{y}_{i-k}$. The overarching goal is to bridge the gap between existing methodologies and the intricate long-term temporal interrelations inherent within sequential data, thereby refining classification outcomes by integrating a comprehensive temporal context.

The problem is formally defined as follows: Given the time-series dataset, we aim to model the conditional probabilities of class transitions using a Bayesian approach and Conditional Probability Tables (CPTs):

$$P(\mathbf{y}_i | \mathbf{y}_{i-1}, \mathbf{y}_{i-2}, \ldots, \mathbf{y}_{i-k})$$

This involves estimating the probability of the current label $\mathbf{y}_i$ given the sequence of previous labels. This probability prediction is then used to refine the class probabilities predicted for the current window by the deep learning model. We should note that although a convolutional neural network (CNN) was used here as the backbone of the deep learning model, our method is independent of the deep learning architecture and any other architecture can be used.

2.2 Proposed Method

As depicted in Fig. 2, our methodology for enhancing time-series classification accuracy integrates a structured workflow. The time-series dataset undergoes standard preprocessing, with data segmented into windows and labeled accordingly. During the training phase, a deep learning model is trained on each window to learn the mapping between data $\mathbf{X}_i$ and labels $\mathbf{y}_i$. At the same time, the Conditional Probability Tables (CPTs) are built. In the inference phase, the deep learning model classifies test windows, returning class prediction probabilities for each window $\hat{\mathbf{y}}_i$, as is common in time series classification. In parallel, our Bayesian model predicts class probabilities $\hat{\mathbf{y}}'_i$ for the same window based on the values of the previous k labels in the sequence of windows, where k is a hyperparameter specifying how many windows in the past the model should consider. Eventually, $\hat{\mathbf{y}}_i$ and $\hat{\mathbf{y}}'_t$ are combined into a final prediction. Each step of this process is explained in detail in the following subsections.

Data Preprocessing: Segmenting continuous time series data into windows of fixed size is a fundamental preprocessing step for training deep learning models, especially for classification tasks. Given a continuous time series $\{x_t\}_{t=1}^{T}$, where x_t denotes the value at time step t and T represents the total number of time steps, the aim is to transform this series into a set of instances that are suitable for the model to learn from. This involves creating either overlapping or non-overlapping windows of a predefined length N, where each window serves as an input instance to the model.

A windowed instance $\mathbf{X}_i$ from the time series can be formulated as $\mathbf{X}_i = (x_i, x_{i+1}, \ldots, x_{i+N-1})$, with N being the

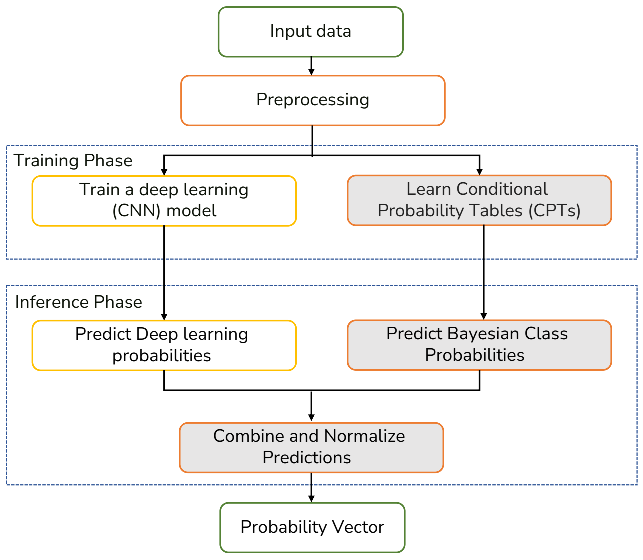
Input data
Preprocessing
Training Phase
Train a deep learning (CNN) model
Learn Conditional Probability Tables (CPTs)
Inference Phase
Predict Deep learning probabilities
Predict Bayesian Class Probabilities
Combine and Normalize Predictions
Probability Vector

Figure 2: Architecture of the model illustrates two phases of the workflow, with each step in the respective phases.

window size, and i ranging from 1 to $T - N + 1$ for non-overlapping windows. The adjustment for i varies for overlapping windows, depending on the step size S chosen for the overlap. For a classification task, the associated label $\mathbf{y}_i$ for each instance $\mathbf{X}_i$ is a vector representing the class of the sequence within the window. In a scenario with C classes, $\mathbf{y}_i$ is a one-hot encoded vector of length C, where each element corresponds to a class, and the element for the correct class is set to 1, while all others are set to 0.

Formally, the segmentation process transforms the continuous time series into a dataset $\{(\mathbf{X}_1, \mathbf{y}_1), (\mathbf{X}_2, \mathbf{y}_2), \ldots, (\mathbf{X}_M, \mathbf{y}_M)\}$, where $M = \lceil (T - N + 1)/S \rceil$ for overlapping windows with step size S. This structured approach enables the efficient utilization of the time series data for model training, facilitating the extraction of meaningful patterns that are predictive of the class labels, thus aligning the input format with the requirements of deep learning models for classification tasks.

Training Phase: The training phase consists of fitting a deep-learning model to find the relationship between the independent variable $\mathbf{X}_i$ and the dependent variable $\mathbf{y}_i$, as well as building the conditional probability tables (CPTs) that model the transition probabilities between classes. The fitting of the deep learning model follows standard practices and any deep learning architecture can be used. Thus, here we focus on explaining the process of learning the creating the CPTs.

Let $Y = \{y_1, y_2, \ldots, y_M\}$ be a sequence of actual class labels, and k the maximum number of previous windows to consider. The goal is to compute a set of Conditional Probability Tables (CPTs), where each CPT is denoted as $P(y_i | y_{i-m}, \ldots, y_{i-1})$ for $1 \le m \le k$ and $m < i \le M$.

The algorithm proceeds as follows:

1. Initialize an empty collection of CPTs, denoted by $\mathcal{C}$.
2. For each label y_i in Y, for $i = 1$ to M:
   (a) Determine the actual number of previous labels to consider, $m = random[1, \min(i - 1, k)]$.
   (b) For the selected m, construct the conditional sequence $S_{i-m}^{i-1} = (y_{i-m}, \ldots, y_{i-1})$.
   (c) Update the count $C(S_{i-m}^{i-1}, y_i)$, which represents the occurrence of label y_i given the sequence S_{i-m}^{i-1}.
3. After processing all labels, for each sequence S and label y in $\mathcal{C}$, convert the counts to probabilities:

$$P(y|S) = \frac{C(S, y)}{\sum_{y'} C(S, y')}$$

where y' iterates over all possible labels given the sequence S.

The resulting collection $\mathcal{C}$ effectively represents the CPTs, where each entry corresponds to the conditional probability of a label given a preceding sequence of labels. This algorithm is described in pseudocode in Algorithm 1.

We should note that the number of previous labels m to consider for each current label y_i is a random integer between 1 and k, and not a constant. That is done for a number of reasons.

1. *Variability in Context Length*: By randomly selecting the length of the preceding sequence of labels to consider, the function introduces variability in the context lengths used to predict the current label. This approach can help the model learn from a diverse set of conditions, rather than being limited to a fixed-length context.
2. *Robustness to Overfitting*: Randomizing the context length can help in preventing the model from overfitting to a particular sequence length. By learning to predict the current label from a variety of previous label lengths, the model may generalize better to unseen data. This technique introduces a form of regularization, as it prevents the model from relying too heavily on specific patterns that only exist for a particular context length.
3. *Efficiency and Exploration*: In scenarios where the optimal context length is unknown, randomly varying m allows the algorithm to explore different context lengths without the need for extensive parameter tuning. This can be especially useful in early stages of model development or when dealing with datasets where the temporal dependencies are not well understood.
4. *Modeling Real-world Uncertainty*: In many real-world scenarios, the relevance of past information may vary unpredictably. By introducing randomness in the selection of m, the function mimics this uncertainty, potentially making the learned CPTs more robust to variations in the importance of historical data.

Inference Phase: During the inference phase, the baseline deep learning model is applied to the test set, taking each window of the dataset as input and producing a probability

Algorithm 1 Learn Conditional Probability Tables (CPTs)

```
Require: $y = [y_1, y_2, \dots, y_M]$ (sequence of class labels), $k$ (maximum number of previous windows)
Ensure: $CPTs$ (Conditional Probability Tables)
 1: Initialize $CPTs$ as an empty dictionary
 2: for $i = 1$ to $M$ do
 3:     $m \leftarrow$ Random integer in $[1, \min(k, i-1)]$ if $i > 1$ else 1
 4:     $currentLabel \leftarrow y[i]$
 5:     $previousLabels \leftarrow (y[\max(1, i-m)], \dots, y[i-1])$
 6:     $CPTs[previousLabels][currentLabel] \leftarrow CPTs[previousLabels][currentLabel] + 1$
 7: end for
 8: for all $(previousLabels, counter)$ in $CPTs$ do
 9:     $total \leftarrow$ sum of all counts in $counter$
10:     for all $label$ in $counter$ do
11:         $CPTs[previousLabels][label] \leftarrow \frac{CPTs[previousLabels][label]}{total}$
12:     end for
13: end for
14: return $CPTs$
```

vector $\hat{\mathbf{y}}_i$, where each probability corresponds to the confidences for each class. At the same time, using the CPTs, we predict the Bayesian posterior label probabilities based on the labels predicted for the previous windows. This process can be described as follows:

Given:

- $\mathcal{P}$, a dictionary of CPTs where each key is a tuple representing a sequence of previous labels and its value is a distribution over possible current labels.
- $L = (l_1, l_2, \dots, l_n)$, a tuple of previous predicted labels.
- C, the total number of distinct classes.

Objective:

- To compute a probability vector $\mathbf{p}$ of length C, where each entry p_c represents the probability of the class c given the sequence of previous labels L.

Procedure:

1. Initialize a vector $\mathbf{p}_{sum}$ of length C with all entries set to 0 and a scalar ω_{sum} also set to 0. These variables will accumulate the weighted probabilities and the total weight, respectively.
2. For each possible subsequence S of L, starting from the last label and extending to include up to the entire sequence L, do the following:
   - Let $S = (l_{n-i+1}, \dots, l_n)$ for i ranging from 1 to $|L|$.
   - If S is a key in $\mathcal{P}$, indicating that we have a CPT for this sequence of labels, then:
     – Retrieve the probability distribution $\mathcal{S}_{prob} = \mathcal{P}[S]$.
     – For each class label c and its associated probability $prob_c$ in $\mathcal{S}_{prob}$, update the accumulator: $\mathbf{p}_{sum}[c] \leftarrow \mathbf{p}_{sum}[c] + prob_c \times |S|$.
     – Update the total weight: $\omega_{sum} \leftarrow \omega_{sum} + |S|$.
3. If $\omega_{sum} > 0$, normalize the accumulated probabilities by the total weight to obtain the final probability vector: $p_c = \frac{\mathbf{p}_{sum}[c]}{\omega_{sum}}$ for each class c.
4. If $\omega_{sum} = 0$, implying that no matching subsequences were found in $\mathcal{P}$, set all entries of $\mathbf{p}$ to $\frac{1}{C}$, distributing the probability equally among all classes.

The output is the probability vector $\mathbf{p}$, where each entry p_c is the weighted average probability of class c given the observed sequence of previous labels L. This algorithm is described in pseudocode in Algorithm 2.

At the training time, it was pointed out that the length of the context window m varies between 1 and k. At inference time, conditional probabilities that are derived from longer contexts (i.e., more labels in the past) are given a higher weight, capturing the fact that a longer sequence of known labels contains a higher certainty in the prediction of future labels than shorter contexts.

Algorithm 2 Predict Bayesian Class Probabilities

```
Require: $\mathcal{P}$ (CPTs as a dictionary), $L$ (previous predicted labels as a tuple), $C$ (number of distinct classes)
Ensure: $\mathbf{p}$ (probability vector of length $C$)
 1: Initialize $\mathbf{w}_{sum} \leftarrow 0$, $\mathbf{p}_{sum} \leftarrow \mathbf{0}$ of length $C$
 2: for $i = 1$ to $|L|$ do
 3:     $S \leftarrow (L[|L| - i + 1], \dots, L[|L|])$ ▷ Subsequence of previous labels
 4:     if $S$ in $\mathcal{P}$ then
 5:         $\mathcal{S}_{prob} \leftarrow \mathcal{P}[S]$
 6:         $w \leftarrow |S|$ ▷ Weight, proportional to subsequence length
 7:         for all $(label, prob)$ in $\mathcal{S}_{prob}$ do
 8:             $\mathbf{p}_{sum}[label] \leftarrow \mathbf{p}_{sum}[label] + prob \cdot w$
 9:         end for
10:         $\mathbf{w}_{sum} \leftarrow \mathbf{w}_{sum} + w$
11:     end if
12: end for
13: if $\mathbf{w}_{sum} = 0$ then return $\left[\frac{1}{C}\right]$ repeated $C$ times
14: else
15:     $\mathbf{p} \leftarrow \left[\frac{p_{sum}}{\mathbf{w}_{sum}}$ for $p_{sum}$ in $\mathbf{p}_{sum}\right]$
16:     return $\mathbf{p}$
17: end if
```

Combining Probabilities from DL and Bayesian Models: The final stage of the inference phase is to combine the probabilities returned by the deep learning model with those returned by the Bayesian model, thus refining the original prediction. This process goes as follows.

Given two probability vectors $\mathbf{d} = [d_1, d_2, \dots, d_C]$ and $\mathbf{b} = [b_1, b_2, \dots, b_C]$, representing the class probabilities predicted by a deep learning model and a Bayesian model respectively, and a weight $\lambda \in [0, 1]$, the objective is to compute a combined probability vector $\mathbf{p} = [p_1, p_2, \dots, p_C]$. The combined probabilities are calculated as a weighted average of the corresponding probabilities from $\mathbf{d}$ and $\mathbf{b}$ for each class c, as follows:

$$p_i = \lambda d_i + (1 - \lambda) b_i, \quad \text{for } i = 1, \dots, C$$

where C is the number of classes.

This process results in $\mathbf{p}$, where each p_i represents the normalized probability of class i, reflecting a consensus prediction between the deep learning and Bayesian models. Algorithm 3 describes this process in pseudocode.

The hyper-parameter λ is defined by the user, and it reflects how much weight we give to the deep learning prediction versus the Bayesian prediction. Under usual circumstances, the deep learning prediction gets a higher weight. In this study, the optimal value of λ was tuned for each dataset using a linear search on the validation set. Usually, a λ value close to 0.8 worked best.

Algorithm 3 Combine and Normalize Predictions

```
Require: $\mathbf{d}$ (probabilities from deep learning model), $\mathbf{b}$ (probabilities from Bayesian model), $\lambda$ (weight for deep learning model probabilities)
Ensure: $\mathbf{p}$ (combined probability vector)
1: Initialize $\mathbf{c}$ to be an empty list of size equal to $\mathbf{d}$
2: for $i = 1$ to $|\mathbf{d}|$ do
3:     $c_i \leftarrow \lambda \cdot d_i + (1 - \lambda) \cdot b_i$     ▷ Calculate weighted average
4:     Update $\mathbf{p}[i]$ with $p_i$
5: end for
6: return $\mathbf{p}$
```

3 Experiments and Results

3.1 Datasets

Our methodology was rigorously tested across several time series datasets, each selected for its unique characteristics and challenges in human activity recognition. These datasets include UniMiB SHAR, UCI HAR, Leotta 2021, and TWristAR.

- **UniMiB SHAR (Micucci, Mobilio, and Napoletano 2017):** Comprises accelerometer data from 30 individuals, capturing 9 distinct activities, suitable for assessing model performance on daily human actions. The sequence length is 151 timesteps, with each sample having 1 channel (total acceleration magnitude). Train (4601 samples), Validation (1454 samples), and Test (1524 samples).
- **UCI HAR Dataset (Jain and Kanhangad 2017):** Features 6 activity types from wearable sensors of 30 volunteers, highlighting its application in wearable technology. The sequence length is 128 timesteps and 4 channels (X, Y, Z, magnitude acceleration). Train (5514 samples), Validation (1838 samples), and Test (2947 samples).
- **Leotta 2021 (Leotta, Fasciglione, and Verri 2021)**: This dataset contains three-axial accelerometer, magnetometer, and gyroscope data recorded from different parts of the body: dominant wrist, hip, and ankle while performing 17 different daily-life activities. The dataset includes data of 8 volunteers. The sequence length is 300 timesteps and 3 channels (X, Y, Z, magnitude acceleration channels). Train (2391 samples), Validation (1167 samples), and Test (1987 samples).
- **TWristAR (Hinkle, Atkinson, and Metsis 2022):** This is a three subject dataset recorded using an e4 wristband. Each subject performed six scripted activities: upstairs/downstairs, walk/jog, and sit/stand. The dataset contains motion (accelerometer) data, temperature, electrodermal activity, and heart rate data. The sequence length is 96 timesteps and 1 channel (total acceleration magnitude). Train (1869 samples), Validation (208 samples), and Test (1091 samples).

3.2 Baseline CNN Model

Architecture

- Input layer: $(n_{\text{steps}}, n_{\text{feats}})$, for n_{steps} time steps and n_{feats} features
- 2 Conv1D layers, 100 filters, kernel size k_{size}, ReLU activation
- Dropout (rate 0.5)
- MaxPooling1D (window: 2)
- Flatten
- 2 dense layers, first 100 units, ReLU, final softmax output

Hyperparameters

- k_{size}: varied for temporal dependencies
- 100 convolutional filters
- Dropout rate 0.5
- Batch size & epochs adjusted per dataset

Training Used Adam, categorical cross-entropy loss.

3.3 Results

We evaluate the performance of our proposed method on four different human activity recognition (HAR) datasets and compare the performance of our model against a baseline CNN model, and against three state-of-the-art time series classification deep learning methods, namely InceptionTime (Ismail Fawaz et al. 2020), Time-Series Transformer (TST) (Zerveas et al. 2021) and LSTNet (Lai et al. 2018). The baseline CNN model is the same model as the one we use in combination with our Bayesian model. InceptionTime is a popular time series classification model that uses an ensemble of deep CNN models inspired by the Inception-v4 architecture, and TST is a transformer-based framework for multivariate time series classification. Finally, LSTNet is a hybrid LSTM-CNN architecture that attempts to learn both short-term and long-term patterns in time series data, which directly relates to our method.

To obtain confidence intervals, each experiment was repeated 10 times, and at each run, each model was re-trained on the same training set but with a different shuffling of the training instances.

The results of our experiments are presented in both Table 1 and the accompanying box plot in Figure 3. The table provides a detailed breakdown of accuracy percentages, standard deviations, and 95% confidence intervals across various datasets, highlighting the consistent performance of our methodology. The box plot visually complements these numerical results, offering insights into the distribution of accuracy values, central tendencies, and potential outliers.

As is evident from the results, our method outperforms the competing methods in all but one dataset. The Leotta

Table 1: This table presents a comprehensive comparison of the average prediction accuracy percentages for the proposed method, baseline CNN, InceptionTime, LSTNet, and TSTPlus models. It includes the standard deviation (Std. Dev.) for the proposed method to indicate result variability and 95% confidence intervals (95% CI) for all methods, providing a statistical perspective on the precision of the accuracy measurements.

Dataset	**Our Method**			**Baseline CNN**		**InceptionTime**		**LSTNet**		**TSTPlus**	
	Acc. %	**Std.**	**95% CI**	**Acc. %**	**95% CI**	**Acc. %**	**95% CI**	**Acc. %**	**95% CI**	**95% CI**	**Acc. %**
UniMiB SHAR	**92.1**	0.90	[90.0, 93.1]	86.4	[84.3, 88.2]	85.0	[83.2, 87.6]	90.4	[88.3, 92.0]	86.1	[85.3, 88.2]
UCI HAR	**82.1**	0.69	[81.1, 82.9]	79.8	[78.2, 81.1]	78.9	[78.1, 79.6]	80.6	[79.0, 81.3]	80.2	[79.4, 81.3]
TWristAR	**78.4**	0.98	[77.5, 80.7]	71.5	[68.4, 75.2]	75.4	[72.1, 77.4]	75.2	[73.1, 76.9]	72.8	[71.2, 74.5]
Leotta	45.0	1.56	[42.3, 47.1]	35.0	[31.1, 36.7]	**46.7**	[45.2, 48.0]	41.9	[40.6, 43.3]	38.4	[37.5, 39.3]

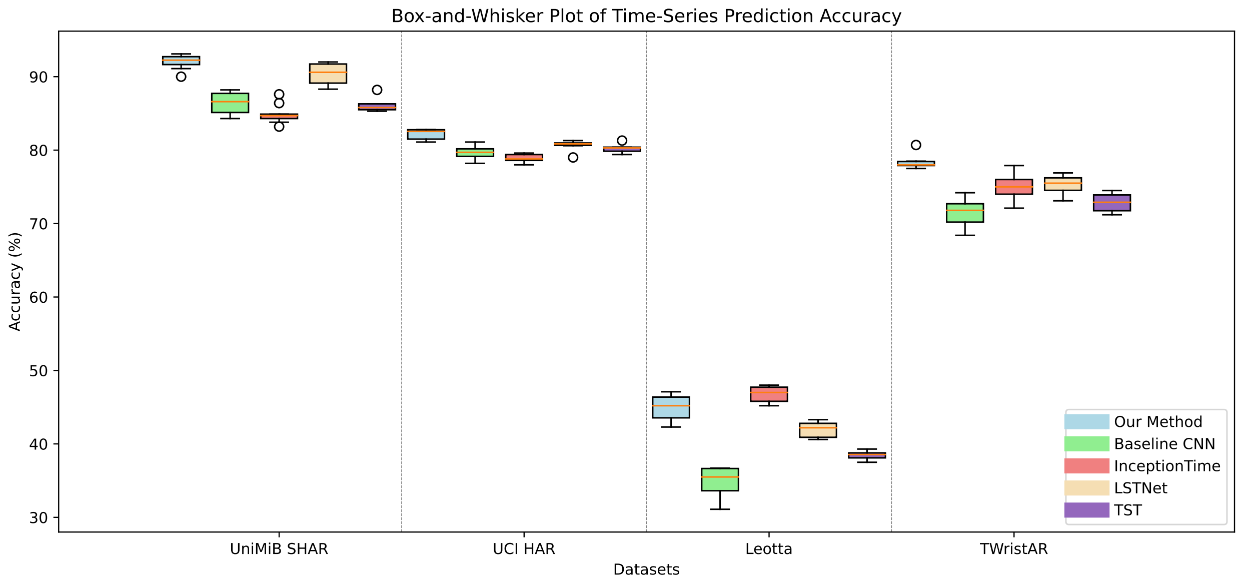

Figure 3: Box Plot visualization of the results from Table 1.

dataset, in which our method comes second overall, produces much lower accuracy for all models. That can be explained by the fact that it includes a much larger set of classes (17) compared to the other datasets involving activities of daily living (ADLs). This dataset also contains more frequent transitions between different classes, which likely diminishes the stabilizing effect of class transition that our method provides.

In three out of the four datasets, we also observe that our method has lower variance and, subsequently, tighter intervals, indicating higher stability across different runs. This, again, is likely the result of the Bayesian prediction adding a stabilizing effect on final predictions.

The results support our hypothesis that modeling temporal context regarding label transitions and past labels, can enhance the prediction accuracy of a model. Most modern deep-learning time series classification methods ignore this fact and treat each window as a separate instance independent of the neighboring instances. Previous attempts at learning long-term dependencies (e.g. LSTNet) have focused on learning long-term patterns from the data rather than the labels. This work has shown that performing inference on the labels can be just as beneficial, and it does not require a deep neural network to learn label transitions, as labels are usually discrete and can be more easily modeled as probability tables.

Again, it should be emphasized that the proposed approach can be added on top of any deep learning model to improve prediction accuracy and stability. In our experiments, it was combined with a basic CNN model, and it not only improved the predictions of the CNN model in every dataset but also outperformed the more advanced time-series classification models in most cases.

4 Conclusion

In this study, we introduced a novel methodology that synergizes Bayesian inference with deep learning to enhance time-series classification, particularly within the realm of human activity recognition. Our approach leverages temporal context to significantly improve predictive accuracy, as evidenced by comprehensive experiments across various datasets. The findings not only validate the effectiveness of integrating temporal context but also set new benchmarks for future research in the field. This research marks a pivotal step towards sophisticated temporal pattern recognition, offering broad implications for both theoretical advancements and practical applications in time-series analysis.

References

Abanda, A.; Mori, U.; and Lozano, J. A. 2019. A review on distance based time series classification. *Data Mining and Knowledge Discovery* 33(2):378–412.

Alzubaidi, L.; Zhang, J.; Humaidi, A. J.; Al-Dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaría, J.; Fadhel, M. A.; Al-Amidie, M.; and Farhan, L. 2021. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data* 8(1):1–74.

Bagnall, A.; Lines, J.; Bostrom, A.; Large, J.; and Keogh, E. 2017. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery* 31:606–660.

Botsch, M. 2023. *Machine learning techniques for time series classification*. Cuvillier Verlag.

de Mattos Neto, P. S.; Cavalcanti, G. D.; Firmino, P. R.; Silva, E. G.; and Nova Filho, S. R. V. 2020. A temporal-window framework for modelling and forecasting time series. *Knowledge-Based Systems* 193:105476.

Hinkle, L. B.; Atkinson, G.; and Metsis, V. 2022. Twristar - wristband activity recognition. `https://zenodo.org/records/5911808`.

Ismail Fawaz, H.; Forestier, G.; Weber, J.; Idoumghar, L.; and Muller, P. A. 2019. Deep learning for time series classification: A review. *Data Mining and Knowledge Discovery* 33(4):917–963.

Ismail Fawaz, H.; Lucas, B.; Forestier, G.; Pelletier, C.; Schmidt, D. F.; Weber, J.; Webb, G. I.; Idoumghar, L.; Muller, P.-A.; and Petitjean, F. 2020. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery* 34(6):1936–1962.

Jain, A., and Kanhangad, V. 2017. Human activity classification in smartphones using accelerometer and gyroscope sensors. *IEEE Sensors Journal* 18(3):1169–1177.

Khan, M.; Wang, H.; Riaz, A.; Elfatyany, A.; and Karim, S. 2021. Bidirectional lstm-rnn-based hybrid deep learning frameworks for univariate time series classification. *The Journal of Supercomputing* 77:7021–7045.

Kirichenko, L.; Radivilova, T.; and Bulakh, V. 2018. Machine learning in classification time series with fractal properties. *Data* 4(1):5.

Lai, G.; Chang, W.; Yang, Y.; and Liu, H. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, 95–104.

Leotta, M.; Fasciglione, A.; and Verri, A. 2021. Daily Living Activity Recognition Using Wearable Devices: A Features-rich Dataset and a Novel Approach.

Micucci, D.; Mobilio, M.; and Napoletano, P. 2017. Unimib shar: A dataset for human activity recognition using acceleration data from smartphones. *Applied Sciences* 7(10):1101.

Wang, Z.; Yan, W.; and Oates, T. 2017. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, 1578–1585. IEEE.

Zerveas, G.; Jayaraman, S.; Patel, D.; Bhamidipaty, A.; and Eickhoff, C. 2021. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2114–2124.

Zhang, Y.; Zhu, Y.; and Zhang, Y. 2020. Enhancing temporal representation learning for time series classification. *IEEE Transactions on Knowledge and Data Engineering* 33(1):1–1.