Multiloss-Based Optimization for Time Series Data Augmentation

Omar Bahri, Peiyu Li, Soukaïna Filali Boubrahimi, Shah Muhammad Hamdi
Department of Computer Science, Utah State University, Logan, UT 84322
Email: {omar.bahri, peiyu.li, soukaina.boubrahimi, s.hamdi}@usu.edu

2023 IEEE International Conference on Big Data (BigData) | 979-8-3503-2445-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/BigData59044.2023.10386614

***Abstract*—Data augmentation plays an important part in the current success of machine learning and deep learning models. In particular, state-of-the-art architectures in the image recognition field include data augmentation modules as an integral part. However, there is still room for progress in the time series domain. In this work, we introduce OptimAug, a novel method for time series data augmentation. We deviate from the current state-of-the-art comprised of random transformations, pattern mixing, generative models, and decomposition methods, to develop the first multiloss-based optimization method. We evaluate our method with its two variants on datasets from the University of California Riverside (UCR) archive and compare it to multiple baseline algorithms from the literature.**

***Index Terms*—Data Augmentation, Time Series Mining**

I. Introduction

The volume of continuously collected and stored time series data has reached staggering levels. However, processing this raw data into clean and labeled datasets requires a lot of time and effort. As pointed out by Iwana et al. [1], the University of California Riverside (UCR) archive [2], one of the largest and most commonly used collections of time series datasets, contains only 12 datasets with a training size higher than one thousand. In addition, time series datasets from real-life domains might suffer from class imbalance due to the rarity of certain events (most likely the desired positive class in the context of classification). For example, the ECG200 dataset from the UCR archive contains 133 normal heartbeats and only 67 myocardial infarctions, and the Earthquakes dataset contains 368 cases and only 93 positive ones. These problems become particularly relevant when training data-hungry deep learning architectures for both classification and forecasting purposes. Therefore, data augmentation approaches aim to increase the size of existing datasets by creating synthetic data samples, allowing trained machine learning models to expand their decision boundary by exploring new input space, increasing their generalization ability, and reducing overfitting.

Data augmentation has proved to be the most effective in the context of image recognition, from simple transformations ingrained in state-of-the-art Convolutional Neural Network (CNN) architectures such as cropping, rotating, mirroring, scaling and color augmenting, to more elaborate autoencoder and Generative Adversarial Networks (GAN) based models. Efforts have also been made to develop data augmentation strategies for time series. According to [1], the current literature can be categorized into the four following main categories: random transformations, similar to the aforementioned image techniques; pattern mixing where multiple instances from the original dataset are combined to create a new one; generative models that attempt to learn the feature distribution; and decomposition that extracts specific features and use them to generate new elements.

All the aforementioned methods aim to generate new samples that are distinct from the original ones and extend the intra-class boundaries of the original samples but are still within the original distribution. However, to our knowledge, no previous effort has adopted a loss-based optimization strategy. In this work, we develop OptimAug, a novel time series data augmentation method that generates new data samples by optimizing the trade-off between these desired criteria. For this purpose, we propose a multi-objective loss function that uses class prototypes or autoencoders to guide the generation process. We show that OptimAug succeeds in generating high-quality augmented data by testing it on a subset of datasets from the UCR archive and comparing it to state-of-the-art time series data augmentation methods from the four established categories. The rest of this paper is organized as follows. In Section II, we formalize the time series data augmentation problem and describe our proposed algorithm. In Section III, we perform an in-depth experimental evaluation, and discuss results. We conclude with a summary in Section IV.

II. Proposed Methodology

A. Problem Definition

Time series data augmentation is performed in order to extend existing datasets and to help machine learning models identify unseen data samples. In particular, samples that fall close to the class boundaries of the original training dataset represent a challenge for both classification and regression models. Therefore, as supported by the literature [1], [3], [4], augmentation methods should add variability to the dataset by creating samples that are distinct from the original ones and that extend the boundaries separating the different dataset classes. In particular, DeVries and Taylor [4] compared the product of using the interpolation operator and the extrapolation operator with their proposed augmentation method and concluded that the data generated using the former is useless and results in performance loss, whereas the data generated using the latter improves the performance of trained classifiers.

In addition, the generated data should be realistic, i.e. it should still respect the class distribution of the original data [1], [3].

In this light, the problem of generating useful time series data samples from original dataset instances can be formulated as follows. Consider a time series dataset $\mathcal{D} = \{TS_1, TS_2, ..., TS_n\}$ such that each sample TS_i is assigned to a class $C_l \in \{C_1, C_2, ..., C_L\}$ and represented by a vector of T time steps $TS_i = [(TS_i)_1, (TS_i)_2, ...(TS_i)_T]$. Let f_l be the probability distribution function of class C_l, i.e., $\{TS_i \mid TS_i \in C_l\} \sim f_l$. A synthetic sample TS_{new} generated from TS_i should fulfill:

(a) $dist(TS_{new}, TS_i) > \epsilon$, where $dist$ is a distance function and $\epsilon > 0$. This ensures that the generated samples introduce variability to the dataset.

(b) $TS_{new} \sim f_l$, i.e., TS_{new} is a sample from f_l. This ensures that the generated samples belong to the original class distribution.

B. Approach

Following the data generation formalism defined above, OptimAug generates a new time series sample TS_{new} from an original dataset instance $TS_i \in \mathcal{D}$ by minimizing a multi-objective loss function L that considers (a) and (b).

$$L = -\alpha L_a + \beta L_b \tag{1}$$

such that $\alpha, \beta \in]0, 1[$ and L_a is the distance between TS_i and TS_{new} computed as the L_1-norm of their difference , with $(TS_{new})_{init} = TS_i$:

$$L_a = ||TS_i - TS_{new}||_{L_1} = \sum_{t=0}^{T} \left|(TS_i)_t - TS_{(new)_t}\right| \tag{2}$$

By minimizing $-L_a$, the objective function L encourages the generation of samples that are distinct from the original ones, hence increasing variability. Concerning L_b, OptimAug implements the two following alternatives.

1) Prototype Loss: A class prototype is a data point that is the most representative of the class elements. OptimAug finds the prototype $\{Pr_1, Pr_2, ..., Pr_L\}$ of each class C_l in $\{C_1, C_2, ..., C_L\}$ by applying Dynamic Time Warping Barycenter Averaging (DBA) [5], [6] to its elements separately. DBA is a global averaging technique that computes the average of a set of sequences. Starting with an initial sequence, DBA computes its squared distances to each sequence in the set. Then, it iteratively modifies the initial sequence to minimize the sum of squared distances. The distance measure used by DBA is Dynamic Time Warping (DTW) defined in Equation 3. DTW uses a dynamic programming approach to calculate the optimal alignment between two sequences s_1 and s_2 of lengths T_1 and T_2 . It creates a two-dimensional matrix D based on Equation 3, where each element represents the accumulated distance between two corresponding points in the sequences. The goal is to find the path through this matrix with the minimum total distance, which represents the best alignment.

$$\begin{aligned} D_{i,j} = dist\,(s_1, s_2) + \min\{D_{i,j-1}, D_{i-1,j}, D_{i-1,j-1}\} \\ \text{s.t} \ : i \in [1, T_1], j \in [1, T_2] \end{aligned} \tag{3}$$

where $dist$ is a distance function, usually the L_1-norm.

Let $TS_i \in C_l$. To encourage the generated TS_{new} to remain within the original class distribution f_l of class C_l, the loss term L_b in OptimAug is set to L_{Pr} defined in Equation 4 as the L_1-norm of the difference between TS_{new} and the prototype Pr_l of C_l:

$$L_{Pr} = ||TS_{new} - Pr_l||_{L_1} = \sum_{t=0}^{T} |(TS_{new})_t - (Pr_l)_t| \tag{4}$$

2) Autoencoder Loss: The second alternative loss term for keeping generated instances within the original class distributions makes use of autoencoders. An autoencoder is a neural network model made of two modules. The first module consists in an encoder network that transforms the input into a more compact latent representation in the form of lower dimensional vector. Then, the second module acts as a decoder by reconstructing it into the original input.

While autoencoders are usually trained to take advantage of the latent space mapping, OptimAug uses autoencoders by considering the reconstruction loss. If possible, separate autoencoders $\{AE_1, AE_2, ..., AE_L\}$ are trained for each class C_l in $\{C_1, C_2, ..., C_L\}$ using its elements only. Then, for $TS_i \in C_l$, TS_{new} is found by optimizing the loss function in Equation 1 where L_b is set to L_{AE} defined in Equation 5 as the L_1-norm of the reconstruction loss of TS_{new} using AE_l as defined in Equation 5.

$$L_{AE} = ||TS_i - AE_l(TS_{new})||_{L_1} = \sum_{t=0}^{T} |(TS_i)_t - AE_l(TS_{new})_t| \tag{5}$$

By minimizing L_{AE}, OptimAug ensures that each generated time series can be accurately reconstructed using the original class autoencoder AE_l, and therefore belongs to the original class data distribution f_l as represented by AE_l.

Algorithm 1 describes the steps involved in creating an augmented sample TS_{new} from an original dataset instance TS_i using OptimAug. In case a class-specific autoencoder is provided, OptimAug uses the autoencoder loss. Otherwise, it defaults to the prototype loss by computing DBA class prototypes as described above. In what follows, we refer to the prototype loss version of the algorithm as OptimAug Proto and to the autoencoder version as OptimAug AE.

III. Experiments

A. Baselines and Implementation Details

We compare the performance of OptimAug to 9 baselines:

- **SMOTE:** [7] interpolates two time series samples (a random one and its nearest neighbor) to generate a new one by computing their difference and multiplying it by a random number between 0 and 1.

Algorithm 1 OptimAug augmentation algorithm

Parameters: α and β
Inputs: Sample TS_i from the dataset $\mathcal{D}$. Class C_l of TS_i. Autoencoder AE_l trained on class l instances (optional).
Output: Augmented sample TS_{new}.

1: Initialize TS_{new} to the original sample TS_i:
$TS_{new} \leftarrow TS_i$
2: **if** AE_l is provided as input **then** use autoencoder loss:
$L_b \leftarrow L_{AE} = \sum_{t=0}^{T} |(TS_i)_t - AE_l(TS_{new})_t|$
3: **else** Get class l prototype and use prototype loss:
$Pr_l \leftarrow DBA(\mathcal{D}_l)$ ▷ **Algorithm 1. in [6]**
where: $\mathcal{D}_l$ is the set of class C_l elements in $\mathcal{D}$
$L_b \leftarrow L_{Pr} = \sum_{t=0}^{T} |(TS_{new})_t - (Pr_l)_t|$
4: **end if**
5: Optimize the loss function L:
$TS_{new} \leftarrow \text{argmin}_{TS_{new}}(-\alpha L_a + \beta L_b)$
where: $L_a = \sum_{t=0}^{T} |(TS_i)_t - TS_{(new)_t}|$
6: **return** TS_{new}

- **ADASYN:** [8] an extension of SMOTE that introduces adaptability to the synthetic sample generation process.
- **guided warping:** [3] warps the original pattern using the alignment function of its DTW distance to a teacher pattern. Random Guided Warping (RGW) selects the teacher randomly from the dataset elements of the same class of origin while Discriminative Guided Warping (DGW) uses a directed one.
- **SuboPtimAl Warped time series geNEratoR (SPAWNER):** [9] uses suboptimal time warping to create new time series. It reduces the flexibility of DTW by forcing the warping path through a random point. The alignment path is then used to generate new patterns.
- **weighted DTW Barycentric Averaging (wDBA):** [10] adapts DBA [5] for data augmentation. This is done using one of three weighting schemes. Since the Average Selected with Distance (ASD) which weights all dataset elements of the original class according to their distance to the original pattern had the best results in the original paper, we use it as our benchmark.
- **TimeGAN:** [11] a Generative Adversarial Networks(GAN) model that combines both encoder-decoder models and adversarial training. It learns the feature distribution by taking advantage of the unsupervised adversarial loss and introducing a supervised loss dictated by the original training data. TimeGAN also develops an embedding model that reduces the high-dimensionality of the adversarial training space by acting as a two-way mapping between features and latent representations.
- **Recurrent Conditional GAN (RCGAN):** [12] a GAN architecture with two recurrent neural networks as the discriminator and generator, conditioned with additional information in order to generate labeled data. The conditioning in RCGAN is done by augmenting the inputs at each time step through concatenation.
- **Time-Conditional Generative Adversarial Network (T-CGAN):** [13] originally proposed for irregularly sampled time series data, it consists in a conditional GAN architecture where the generator module is represented by a deconvolutional network and the discriminator module by a convolutional network.

We adopted the implementations provided by the authors of [1][1], [12][2], and [13][3]. For TimeGAN, we used the implementation available in YData Synthetic Python package[4], and for SMOTE and ADASYN, we used the implementations in the imbalanced-learn Python package[5]. We used the default configurations for all algorithms. The code for OptimAug is available in the project's GitHub repository[6]. In theory, since all the loss terms described in Section II are fully differentiable, any first-order optimization algorithm should work for OptimAug. In the following experiments, we used the Adam optimizer. To test OptimAug AE, we trained simple, shallow class autoencoders consisting of two Long Short-Term Memory (LSTM) layers, with the second layer outputting a sequence with the original time series length, followed by the same dense layer applied to each time step individually. We used the Adam optimizer, the Scaled Exponential Linear Unit (SELU) activation function for the LSTM layers, and L2 regularization. The only tuning we performed was for the learning rate and regularization, with the only purpose of avoiding training problems such as the exploding gradient.

B. Evaluation Criteria

1) Performance Gain: To assess the performance of OptimAug in comparison to the nine time series data augmentation baselines, we compare the classification performance gained from augmenting the original datasets to double the original size with each method. For this purpose, we use two state-of-the-art machine learning classifiers: (a) Residual Network (ResNet), a deep neural network that was first introduced for time series classification by Wang et al. [14] and that achieved state-of-the-art performance for classifying UCR datasets [15] and (b) RandOm Convolutional KErnel Transform (ROCKET) which had the highest rank and the fastest running time among all models benchmarked in an extensive time series classification survey [16].

a) ResNet:: First, we split the original dataset into training, test, and validation splits and train a ResNet model for 1500 epochs using the Adam optimizer, following the original configuration in [14]. During training, we save the weights of the model that performed the best on the validation split and use them to evaluate the classification performance on the test set. For more robustness, we train three different models with different initial weights for each dataset and report the

[1]https://github.com/uchidalab/time_series_augmentation
[2]https://github.com/ratschlab/RGAN
[3]https://github.com/gioramponi/GAN_Time_Series
[4]https://github.com/ydataai/ydata-synthetic
[5]https://imbalanced-learn.org/stable/
[6]https://github.com/omarbahri/OptimAug

mean scores. Since the dataset might not be balanced across all classes, we adopt the f1-score as classification measure.

Then, for each data augmentation method, we combine the original training sets with the augmented data, and repeat the above to train three new models. Finally, we compare the mean f1-scores of the new models on the original test sets to the previous ones to assess the of each augmentation method.

b) ROCKET:: We repeat the same procedure described for ResNet. Since ROCKET training time is considerably lower, we train each model ten times instead of three and consider the mean f1-score for each method.

2) Novelty Detection: The augmented data samples have to respect the original dataset distribution. To ensure that OptimAug fulfills this condition, we introduce an evaluation criterion in the form of outlier detection. For this purpose, we adopt three novelty detection methods and apply them to OptimAug and the baselines. The first method is the Local Outlier Factor (LOF) [17] which computes the local density deviation of every sample with reference to its neighboring data points. The data samples with low density values are detected as outliers. The second method is Isolation Forest (IF) [18] which focuses on the distance of each data sample to the entire dataset. The third method is the One Class Support Vector Machine (OC-SVM) [19] method that learns a hypersphere encompassing all original instances and flags samples that lie outside as outliers. We apply each method to the data generated by the augmentation baselines in this study and rank them according to the percentage of detected outliers.

C. Illustrative Example: Cylinder-Bell-Funnel (CBF) Dataset

The Cylinder-Bell-Funnel (CBF) is a dataset from the UCR archive that consists of simulated time series with 128 time steps divided across three classes and split into a training set of size 30 and a test set of size 900. The time series are generated as standard normal noise plus a different offset term for each class. We select CBF as a starting point for the evaluation of OptimAug because of its simplicity and small number of samples in the training set.

1) Performance Gain: As this is the first dataset we apply OptimAaug to, we used it to tune the two hyperparameters α and β based on the ResNet performance gain. For this purpose, we performed a non-exhaustive grid search and settled on $\alpha = 10^{-10}$ and $\beta = 1.1$ or $\beta = 10^{-6}$ for OptimAug Proto loss or OptimAug AE respectively. For the rest of the experiments with different datasets in this paper, we kept the same parameters values.

Given that the mean f1-score using ResNet on the original CBF dataset –without data augmentation– is already high with a value of 99.61%, the main purpose of this test is to ensure that the data generated is not radically different from the original dataset, which might lead to significant performance loss. After adding the data samples generated using OptimAug Proto, the mean f1-score slightly increased to 99.67%, while Optimaug AE resulted in an non-significant decrease to 99.41%. Table I shows that the only augmentation method that significantly affects the model performance is RCGAN. On the other hand, the data generated by DGW achieved the highest increase with 99.85%.

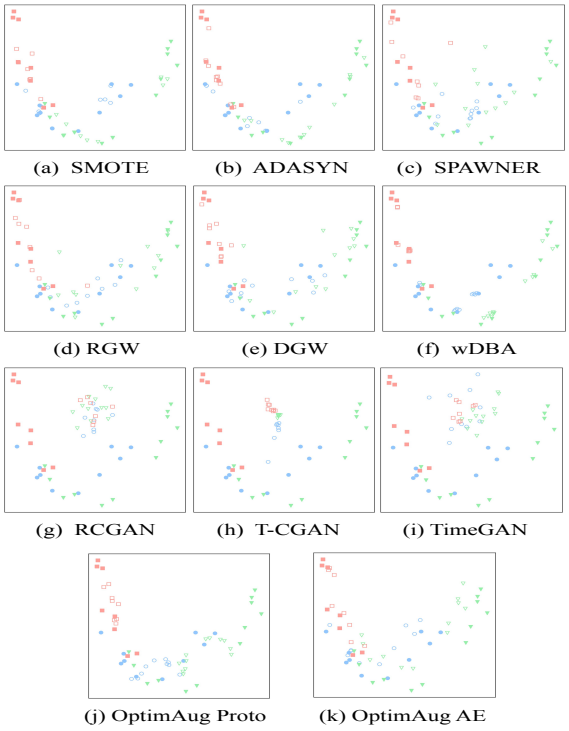

Fig. 1: Visualizing the distributions of augmented data samples (unfilled shapes) compared to the original ones (filled shapes) on the PCA space for the CBF dataset. Each color and shape represents a different class.

2) Novelty Detection: As shown in Table I, the percentages of instances detected as outliers by the three novelty detection methods in the samples generated by OptimAug are low compared to the data produced by the baselines. In fact, OptimAug Proto is the only method that generated data without outliers according to all the benchmarks. On the other hand, 4.67% of Optimaug AE's instances have been flagged by the IF algorithm, a percentage slightly higher than SMOTE's, ADASYN's, and wDBA's, but still a good result given that these methods rely on interpolation and generate samples close to the original ones (as we discuss in the next section).

Table I also shows that, consistent with the previous performance gain results, the data generated by RCGAN is largely out of distribution. In addition, T-CGAN, TimeGAN, and to a certain extent RGW, have also generated a considerable number of outlier data points. This confirms the importance of the novelty detection test to identify unrealistic data regardless of whether or not it benefits classification or prediction.

3) Visualizing Data Distributions: We use Principal Component Analysis (PCA) to project the distributions of the datasets generated by OptimAug and the different baselines on a 2D space and visualize them in Fig 1. The filled shapes represent the original data and the unfilled shapes represent the generated samples. The elements of each class are represented by a unique color and shape.

TABLE I: Mean ResNet F1-scores and percentage (%) of generated instances detected as outliers by the novelty detection methods (CBF dataset)

Method	No Aug	SMOTE	ADASYN	wDBA	SPAWNER	RGW	DWG	RCGAN	T-CGAN	TimeGAN	OptimAug Proto	Optimaug AE
F1-Score	99.61	99.74	99.37	99.78	98.93	99.19	99.85	98.93	97.2	98.89	99.67	99.41
LOF	N/A	0.00	0.00	0.00	0.00	0.00	0.00	21.00	0.00	0.00	0.00	0.00
IF	N/A	1.67	1.29	0.67	7.33	7.67	5.33	57.08	58.21	72.00	0.00	4.67
OC-SVM	N/A	3.00	0.00	0.00	7.00	30.00	13.00	50.00	50.00	33.00	0.00	0.00

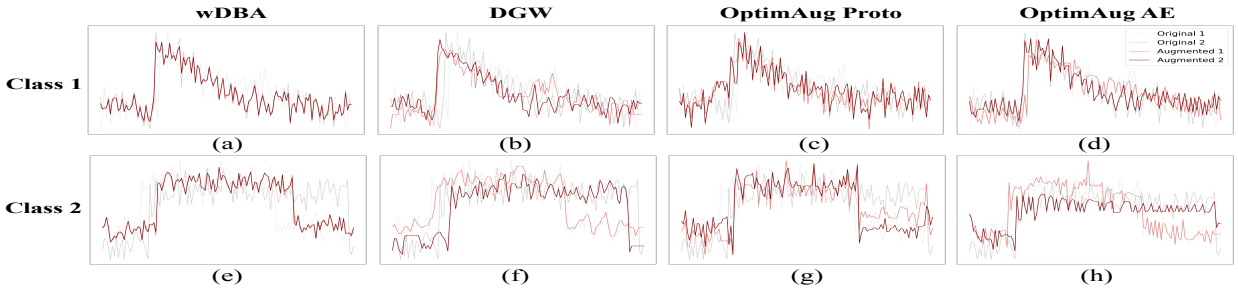

Fig. 2: Visually inspecting augmented data samples generated from two different classes of the CBF dataset using wDBA in (a) and (c), DGW in (b) and (f), Optimaug Proto in (c) and (g), and Optimaug AE in (d) and (h).

By looking at Fig. 1.g, Fig. 1.h, and Fig. 1.i, it is clear that the samples generated by RCGAN, T-CGAN, and TimeGAN are complete outliers, confirming the previous novelty detection results. As to SMOTE, ADASYN, and wDBA, the data generated is too similar and close to the original. The reason is that the three methods leverage direct interpolation between dataset samples to create new ones. Therefore, as shown in, Fig. 1.a, Fig. 1.b, and Fig. 1.f, they do not significantly increase the variability and diversity of the dataset. This also explains their good performance in the novelty detection test (Table I). SPAWNER introduces more variability to the dataset. However, a considerable number of samples generated are out-of-distribution (at the center of the 'U' outlined by the dataset). RGW, DGW, and OptimAug do a good job in extending the intra and inter-class boundaries while respecting the overall dataset distribution. However, RGW and DGW occasionally generate data points outside of the original class distribution and sometimes totally belonging to other classes (e.g. the green points close to the red area for RGW and DGW, and the red point in the blue area for RGW). Therefore, OptimAug generates more diverse samples, introduces more variability to the dataset, extends the class boundaries in different directions while remaining within the respective class distributions, and should in theory be more beneficial for classification tasks.

4) Visualizing Augmented Data Samples: Finally, we visually inspect a few augmented samples from two different classes of the dataset and plot them in Fig.2. In each subfigure, the two gray plots represent original dataset instances and the two red ones represent the samples generated from them. We compare OptimAug's data to wDBA's and DGW's, since they achieved the best performance in the novelty detection test and performance gain test respectively. Fig 2. serves two purposes. Firstly, it shows that the samples generated by OptimAug Proto (Fig. 2.c and Fig. 2.g) and OptimAug AE (Fig. 2.d and Fig. 2.h) look (1) realistic, i.e. visually comparable to the original instances from the same class, (2) different from the original instances, thus introducing variability, and (3) different from each other. Secondly, it confirms previous conclusions such as the fact that wDBA produces similar data instances (the augmented data plots in Fig. 2.a and Fig. 2.e are overlapping) and that DGW occasionally generates outliers (notice the horizontal segment in the middle of Augmented 1 in Fig. 2.b).

D. Evaluation on Different Time Series Datasets

In addition to the CBF dataset, we experiment on 15 more datasets from the UCR time series classification archive [2], spanning eight domains: simulated, traffic, spectro, device, image, EOG, motion, and sensor data, to evaluate OptimAug and compare it to the baseline algorithms. Due to space restrictions, we only present the average ranks in Table II. The detailed results and a summary of the datasets' statistics can be found in our project website[7].

a) ResNet: Table II shows the average rank of each augmentation method, in addition to the performance of the original dataset (without data augmentation). While most algorithms have resulted in scores higher than the original ones on average, SPAWNER, RCGAN, and T-CGAN have actually resulted in significant performance loss. OptimAug Proto and OptimAug AE achieved the highest average performance gain with average ranks of 3.63 and 3.56 respectively. The dataset that benefited the most from the augmentation process was Herring with a performance gain of 33%, from 53.87% to 71.43%, which was achieved by OptimAug Proto[7].

b) Rocket: Table II shows that the Rocket models did not benefit from the data augmentation process as much as the ResNet models. By examining the average f1-scores[7], it is interesting to note that seven out of the nine time series augmentation baselines have in fact hurt the classification performance of the original datasets. Besides OptimAug, only

[7]https://sites.google.com/view/OptimAug/home

TABLE II: Performance gain and outlier detection average ranks.

	No Aug	SMOTE	ADASYN	wDBA	SPAWNER	RGW	DGW	T-CGAN	RCGAN	TimeGAN	Optimaug Proto	OptimAug AE
ResNet Performance Gain	5.94	5.31	5.44	4.94	6.19	5.63	4.44	8.25	5.88	5.69	3.63	**3.56**
ROCKET Performance Gain	5.13	4.94	4.56	6.44	7.06	5.75	7.06	8.50	8.00	7.25	**3.94**	4.36
LOF Outlier Detection	N/A	1.56	2.06	**1.31**	5.56	5.31	6.19	6.73	7.38	7.31	2.38	3.31
IF Outlier Detection	N/A	3.44	4.06	**2.00**	6.06	6.81	7.63	7.75	8.06	8.00	3.19	6.00
OC-SVM Outlier Detection	N/A	1.69	**1.44**	1.75	6.13	7.56	7.88	8.27	8.44	7.31	2.63	4.56

SMOTE and ADASYN resulted in performance gain on average. OptimAug Proto had the highest rank with a value of 3.94, followed by OptimAug AE with 4.38.

Table II also shows that the augmentation methods that rely on interpolation to generate new samples (i.e. SMOTE, ADASYN, and wDBA) produce the lowest amount of outliers. As we discussed in section III.C, this stems from the fact that these methods do not significantly increase the variability of the datasets, and neither do they extend the inter- and intra-class boundaries. OptimAug Proto and OptimAug AE rank right after the three aforementioned methods according to the three novelty detection algorithms. Therefore, OptimAug is the algorithm that introduces the most variability with the minimum number of outliers.

E. OptimAug Proto or OptimAug AE?

Following the results of the previous experiments, it seems that OptimAug Proto and OptimAug AE perform at a similar level. Therefore, when should either of them be used? Going back to the mean f1-scores[8] and focusing on the Computers and Herring datasets, we note that OptimAug AE outperforms OptimAug Proto on the former and vice-versa on the latter. Upon inspecting the class autoencoders used by OptimAug AE, we noticed that the Computers ones produce significantly more accurate reconstructions of the test set instances compared to the Herring autoencoders. One apparent contributing factor is the higher size of the Computers training set. Thus, we recommend using OptimAug AE when accurate class autoencoders are available. In addition, OptimAug Proto generates fewer outliers[8]. This can be explained by the fact that we only trained shallow class autoencoders with minimal hyperparameter tuning.

IV. Conclusion

OptimAug is a novel time series data augmentation method that optimizes a multi-objective loss function to generate new data samples. To our knowledge, this is the first effort to use a simple optimization-based algorithm for the time series augmentation task. Using several datasets from the UCR time series classification archive, we showed that OptimAug introduces variability to the original data space and extends the boundaries within and between classes while respecting the original class distributions, proving its superiority to nine state-of-the-art baseline methods.

Acknowledgments This project has been supported in part by funding from GEO Directorate under NSF awards #2204363, #2240022, and #2301397 and the CISE Directorate under NSF award #2305781.

[8] https://sites.google.com/view/OptimAug/home

References

[1] B. K. Iwana and S. Uchida, "An empirical survey of data augmentation for time series classification with neural networks," *PLOS ONE*, vol. 16, no. 7, p. e0254841, jul 2021.

[2] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, "The UCR Time Series Archive," oct 2018.

[3] B. K. Iwana and S. Uchida, "Time Series Data Augmentation for Neural Networks by Time Warping with a Discriminative Teacher," apr 2020.

[4] T. DeVries and G. W. Taylor, "Dataset Augmentation in Feature Space," feb 2017.

[5] F. Petitjean, A. Ketterlin, and P. Gançarski, "A global averaging method for dynamic time warping, with applications to clustering," *Pattern Recognition*, vol. 44, no. 3, pp. 678–693, mar 2011.

[6] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen, and E. Keogh, "Dynamic Time Warping Averaging of Time Series Allows Faster and More Accurate Classification," *Proceedings - IEEE International Conference on Data Mining, ICDM*, vol. 2015-January, no. January, pp. 470–479, jan 2014.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, jun 2002.

[8] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *Proceedings of the International Joint Conference on Neural Networks*, 2008, pp. 1322–1328.

[9] K. Kamycki, T. Kapuscinski, and M. Oszust, "Data Augmentation with Suboptimal Warping for Time-Series Classification," *Sensors*, vol. 20, no. 1, p. 98, dec 2019.

[10] G. Forestier, F. Petitjean, H. A. Dau, G. I. Webb, and E. Keogh, "Generating synthetic time series to augment sparse datasets," in *Proceedings - IEEE International Conference on Data Mining, ICDM*, vol. 2017-Novem, dec 2017, pp. 865–870.

[11] J. Yoon, D. Jarrett, and M. Van Der Schaar, "Time-series Generative Adversarial Networks," Tech. Rep., 2019.

[12] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs," jun 2017.

[13] G. Ramponi, P. Protopapas, M. Brambilla, and R. Janssen, "T-CGAN: Conditional Generative Adversarial Network for Data Augmentation in Noisy Time Series with Irregular Sampling," nov 2018, arXiv:1811.08295.

[14] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2017-May, jun 2017, pp. 1578–1585.

[15] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, jul 2019.

[16] A. Pasos 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, pp. 401–449, 2021.

[17] M. M. Breuniq, H. P. Kriegel, R. T. Ng, and J. Sander, "LOF," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 93–104, may 2000.

[18] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation forest," *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 413–422, 2008.

[19] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the Support of a High-Dimensional Distribution," *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, jul 2001.