High-Performance Time Series Augmentation Library

Interim Presentation

Aryamaan Basu Roy Tejas Pradhan Felix Kern

17th Jun, 2025

RWTH AACHEN UNIVERSITY

Introduction

1. Time Series...What?
   - Finite Sequence
   - Time ordered indexing
   - Discrete Timestamps
   - Overlapping/Non-Overlapping

Introduction

1. Time Series...What?
   - Finite Sequence
   - Time ordered indexing
   - Discrete Timestamps
   - Overlapping/Non-Overlapping

2. Why Augment?
   - Improve Model Generalization
   - Granular Data Analysis
   - Simulate Real-World Noise
   - Overlapping/Non-Overlapping

Insights From The Paper (Wen et al. 2020)

1. Time Domain Transformations
   - Computationally Cheap
   - Good baseline

Insights From The Paper (Wen et al. 2020)

1. Time Domain Transformations
   - Computationally Cheap
   - Good baseline

2. Frequency Domain Transformations
   - Provides Global Structure
   - Great for Anomaly Detection

Insights From The Paper (Wen et al. 2020)

1. Time Domain Transformations
   - Computationally Cheap
   - Good baseline

2. Frequency Domain Transformations
   - Provides Global Structure
   - Great for Anomaly Detection

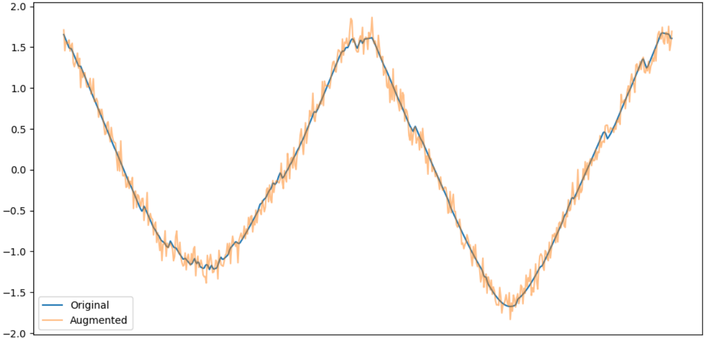
3. Impact
   - 2% Classification Accuracy Improvement
   - 20-30% Improvement - Anomaly Detection F1 Score
   - Reduced MASE error - Forecasting Use Cases

Project Plan

1. Implementation of core time series augmentation methods in Rust
   - Basic transformations

Basic Transformations (Um et al. 2017)

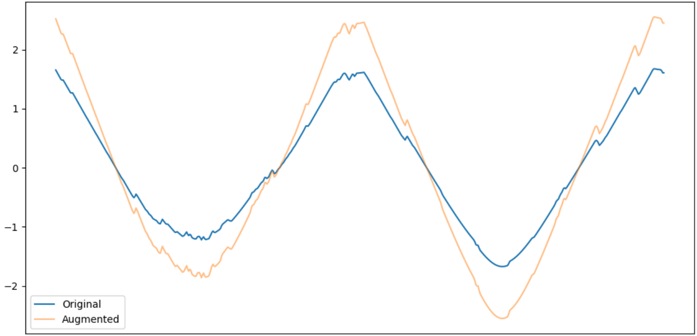
Jittering: Add white noise to the data



Parameters: σ of Gaussian noise

Basic Transformations (Um et al. 2017)

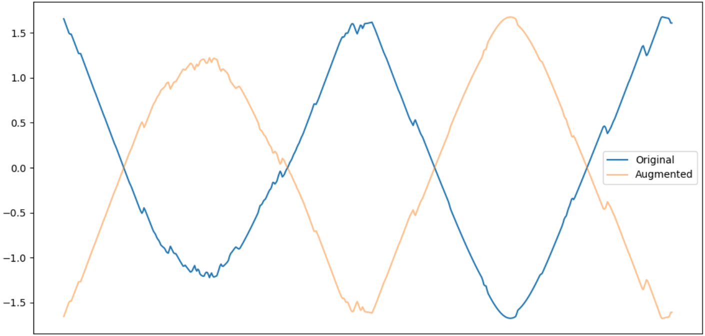
Scaling: Scale data points by random factor



Parameters: Min and max of random factor

Basic Transformations (Um et al. 2017)

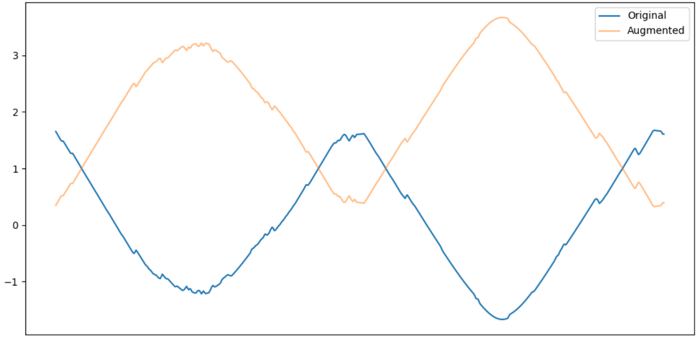
Rotation: "Rotate" data 180° around an anchor



Parameters: Anchor value

Basic Transformations (Um et al. 2017)

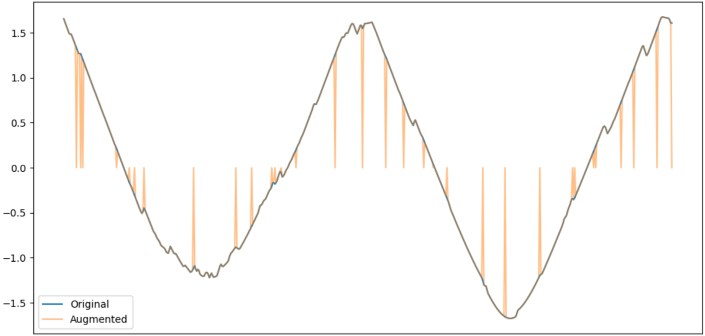
Rotation: "Rotate" data 180° around an anchor



Parameters: Anchor value

Basic Transformations (tsaug)

Drop: Drop percentage of data points



Parameters: Percentage and default value

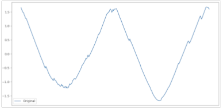
Basic Transformations (tsaug)

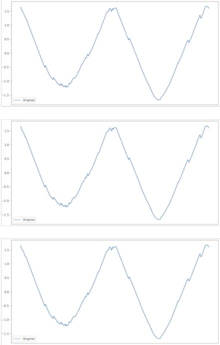
Special Transformations:

Basic Transformations (tsaug)

Special Transformations:

Repeat: Repeat each series in the dataset *n* times



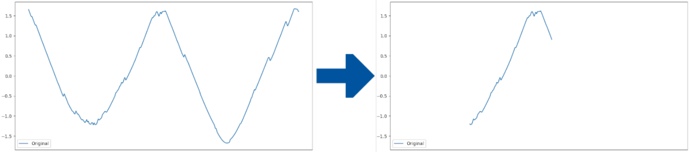


Basic Transformations (tsaug)

Special Transformations:

Repeat: Repeat each series in the dataset *n* times

Crop: Crop each time series into a random continuous slice of specified size

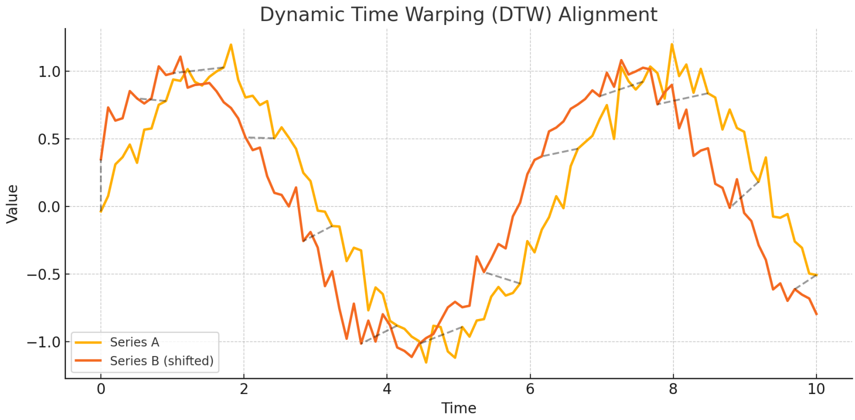


Project Plan

1. Implementation of core time series augmentation methods in Rust
   - Basic transformations
   - Time warping techniques

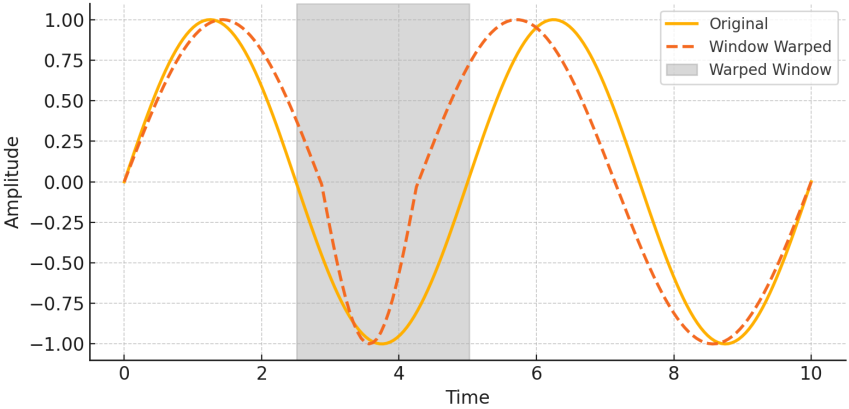
Time Warping

Dynamic Time Warping (DTW): Finds optimal warping path to generate different temporal variations.



Time Warping

Window Warping: Select a random window in the time series and stretch or compress it



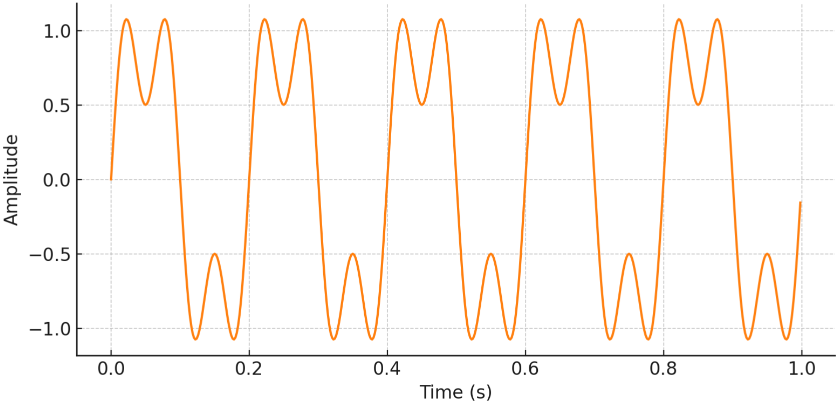
Project Plan

1. Implementation of core time series augmentation methods in Rust
   - Basic transformations
   - Time warping techniques
   - Frequency-domain transformations

Frequency-Domain Transformations

Fast Fourier Transform (FFT): Converts a time-domain signal into its frequency components

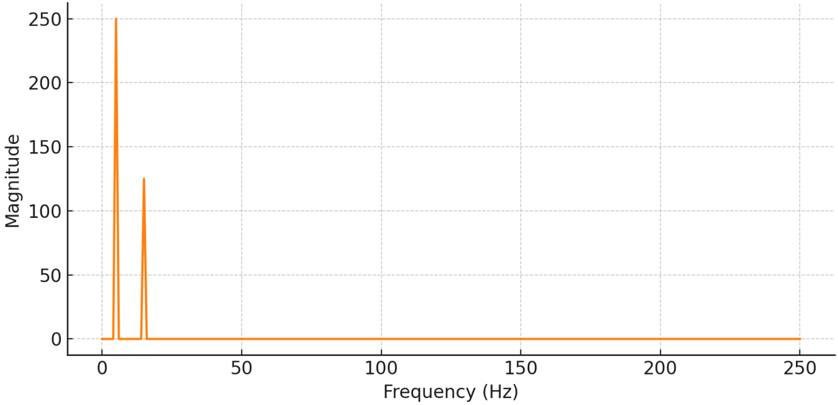
- Time Domain



Frequency-Domain Transformations

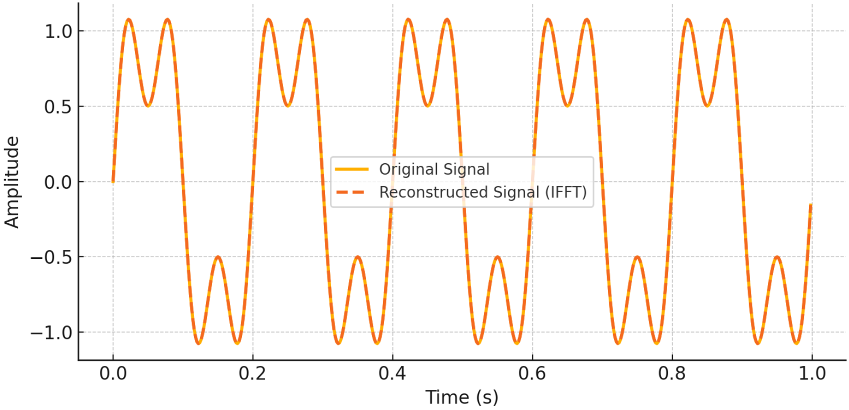
Fast Fourier Transform (FFT): Converts a time-domain signal into its frequency components

- Frequency Domain



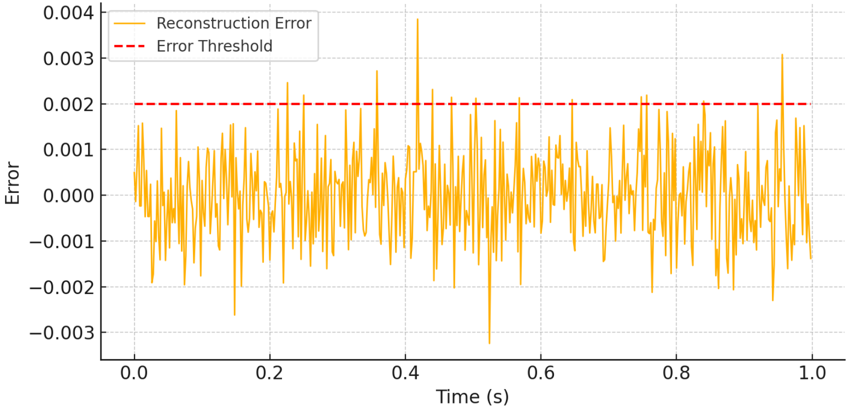
Frequency-Domain Transformations

Inverse Fast Fourier Transform (IFFT): Converts frequency-domain data back to the time domain.



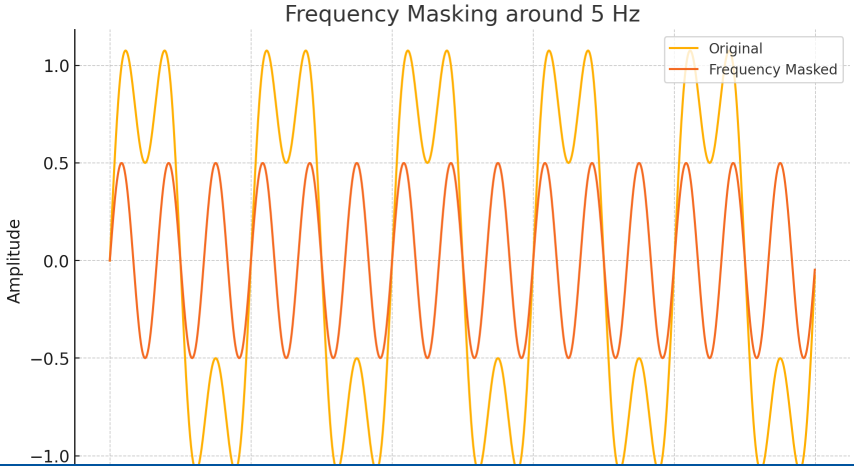
Frequency-Domain Transformations

Inverse Fast Fourier Transform (IFFT): Reconstruction error is negligible (below numerical threshold), ensuring lossless transform.



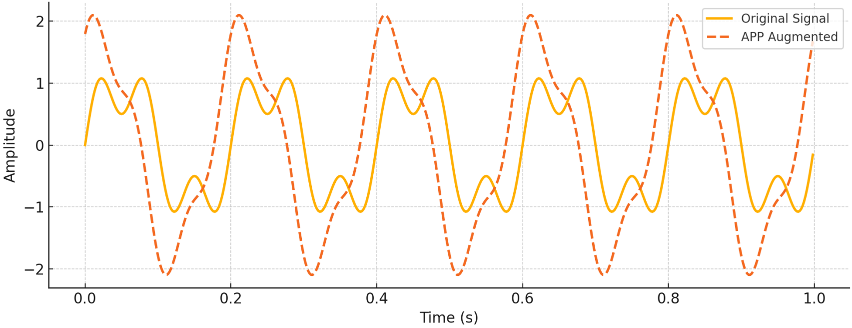
Frequency-Domain Transformations

Frequency Masking: Randomly zero out contiguous FFT bins around a center frequency to simulate narrowband interference or dropout in sensors.



Frequency-Domain Transformations

Amplitude & Phase Perturbation (APP): Add small Gaussian noise to each bin's magnitude and phase to introduce realistic spectral jitter while preserving overall structure



Project Plan

1. Implementation of core time series augmentation methods in Rust
    - Basic transformations
    - Time warping techniques
    - Frequency-domain transformations
    - Noise injection methods

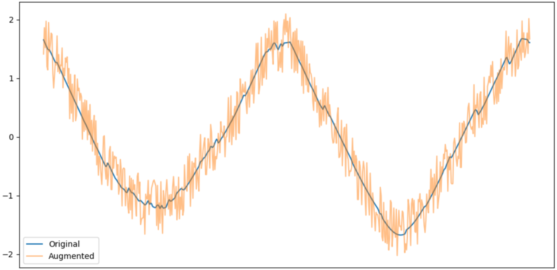
Noise Injection

AddNoise augmenter that supports different kinds of noise (according to Wen et al. 2020):

Noise Injection

AddNoise augmenter that supports different kinds of noise (according to Wen et al. 2020):

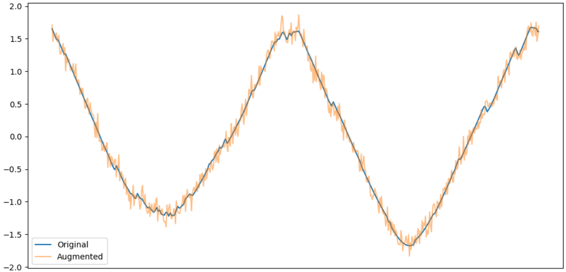
- Uniform



Noise Injection

AddNoise augmenter that supports different kinds of noise (according to Wen et al. 2020):

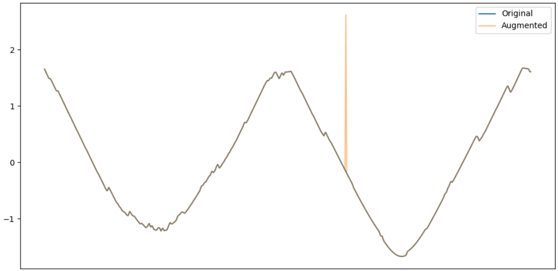
- Uniform
- Gaussian (like jittering)



Noise Injection

AddNoise augmenter that supports different kinds of noise (according to Wen et al. 2020):

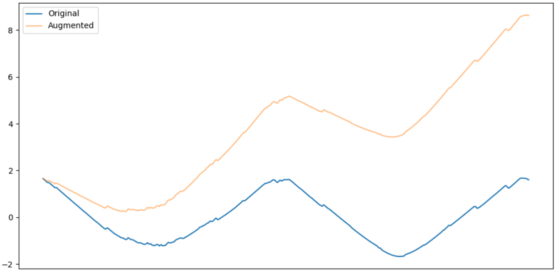
- Uniform
- Gaussian (like jittering)
- Spike



Noise Injection

AddNoise augmenter that supports different kinds of noise (according to Wen et al. 2020):

- Uniform
- Gaussian (like jittering)
- Spike
- Slope



Project Plan

2. Develop a Python-friendly API that enables:

Project Plan

2. Develop a Python-friendly API that enables:
   - Single-method augmentation calls

Project Plan

2. Develop a Python-friendly API that enables:
   - Single-method augmentation calls
   - Composable augmentation pipelines

Implementation - Rust

Every augmenter implements the Augmenter trait:

```
pub trait Augmenter {
    fn augment_dataset(&self, input: &mut Dataset) {
        input.features
            .iter_mut()
            .for_each(|x| self.augment_one(x));
    }

    fn augment_one(&self, x: &mut [f64]);
}
```

Implementation - Rust

⇒ Allows us to create "recursive" augmenters:

```rust
pub struct ConditionalAugmenter {
    inner: Box<dyn Augmenter>,
    p: f64,
}

pub struct AugmentationPipeline {
    augmenters: Vec<Box<dyn Augmenter>>,
}
```

Implementation - Rust

Example pipeline:

```rust
let pipeline = AugmentationPipeline::new()
    + Repeat::new(10)
    + Crop::new(100)
    + ConditionalAugmenter::new(
        AddNoise::new(
            NoiseType::Slope,
            Some((0.01, 0.02)),
            None,
            None
        ),
        0.25
    )
    + Jittering::new(0.1);

pipeline.augment_dataset(&mut dataset);
```

Implementation - Python

- Python bindings using PyO3

Implementation - Python

- Python bindings using PyO3
- In seperate package to Rust library

Implementation - Python

- Python bindings using PyO3
- In seperate package to Rust library
- We bind the struct Dataset to pass the data around

Implementation - Python

- Python bindings using PyO3
- In seperate package to Rust library
- We bind the struct Dataset to pass the data around
- Augmenter structs are exposed to Python as classes

Implementation - Python

- Python bindings using PyO3
- In seperate package to Rust library
- We bind the struct Dataset to pass the data around
- Augmenter structs are exposed to Python as classes
- Limitations with PyO3 don't allow for the same architecture

Implementation - Python

PyO3 makes the binding of the recursive augmenters difficult, so these are fully written in Python for now:

```python
class ConditionalAugmenter:
    def __init__(self, augmenter, probability): ...
    def augment_dataset(self, dataset: Dataset): ...
    def augment_one(self, x): ...

class AugmentationPipeline:
    def __init__(self): ...
    def __add__(self, other): ...
    def augment_dataset(self, dataset: Dataset): ...
    def augment_one(self, x): ...
```

Implementation - Python

Full python example:

```
dataset = pf.Dataset(features, labels)

pipeline = (pf.AugmentationPipeline()
            + pf.Repeat(10)
            + pf.Crop(100)
            + pf.ConditionalAugmenter(
                pf.AddNoise(
                    pf.NoiseType.Slope,
                    bounds=(0.01, 0.02)
                ),
                0.25
            )
            + pf.Jittering(0.1))

pipeline.augment_dataset(dataset)
```

Project Plan

2. Develop a Python-friendly API that enables:
   - Single-method augmentation calls
   - Composable augmentation pipelines

Project Plan

2. Develop a Python-friendly API that enables:
   - Single-method augmentation calls
   - Composable augmentation pipelines
   - Batch processing capabilities

Project Plan

2. Develop a Python-friendly API that enables:
    - Single-method augmentation calls
    - Composable augmentation pipelines
    - Batch processing capabilities
    - Parallel exection options

Project Plan

3. Analyse performance against python library tsaug
   - Execution time benchmarks
   - Memory usage comparisons
   - Quality assessment of augmented data

References

Wen, Qingsong et al. (2020). "Time series data augmentation for deep learning: A survey". In: *arXiv preprint arXiv:2002.12478*.

Um, Terry T et al. (2017). "Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks". In: *Proceedings of the 19th ACM international conference on multimodal interaction*, pp. 216–220.