

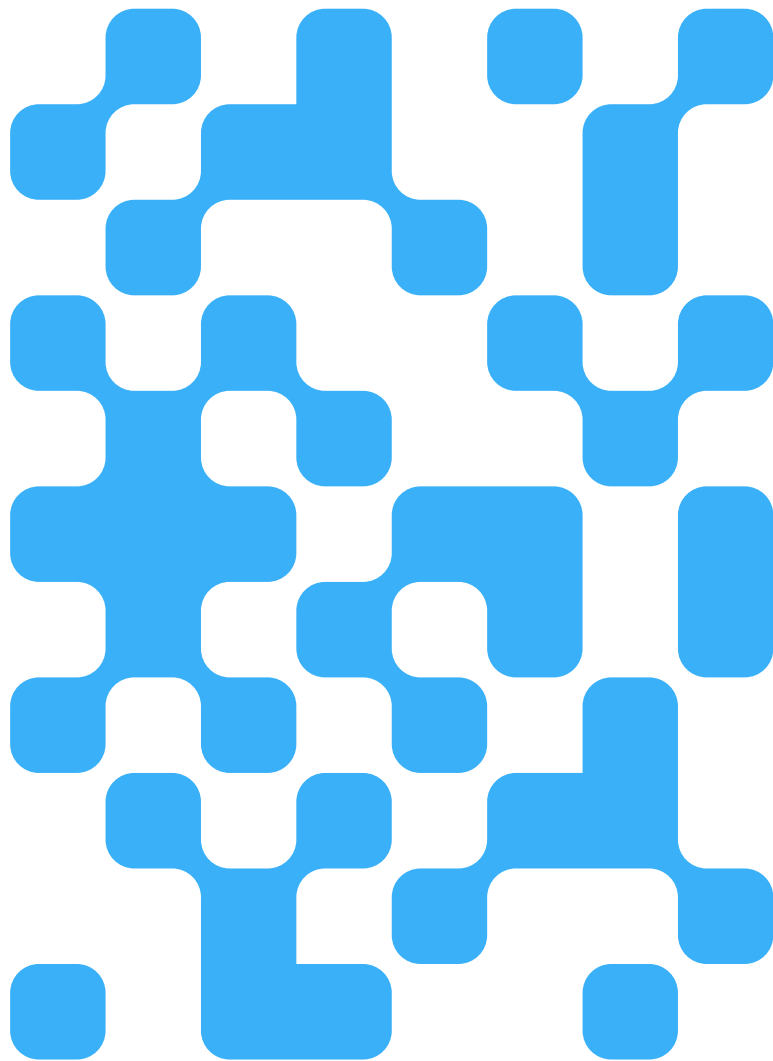


Machine Learning

2024 (ML-2024)

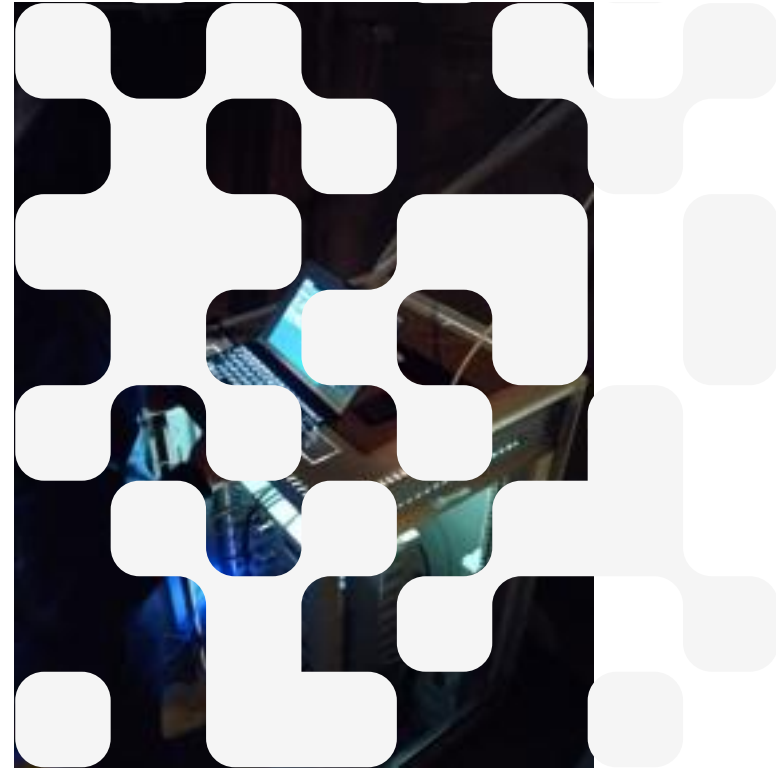
Lecture 15. Temporal and spatial signals analysis

by Alexei Valerievich Kornaev, Dr. habil. in Eng. Sc.,
Researcher at the RC for AI, Assoc. Prof. of the Robotics and CV
Master's Program, [Innopolis University](#)
Researcher at the RC for AI, [National RC for Oncology n.a. NN Blohin](#)
Professor at the Dept. of Mechatronics, Mechanics, and Robotics,
[Orel State University](#)



Agenda

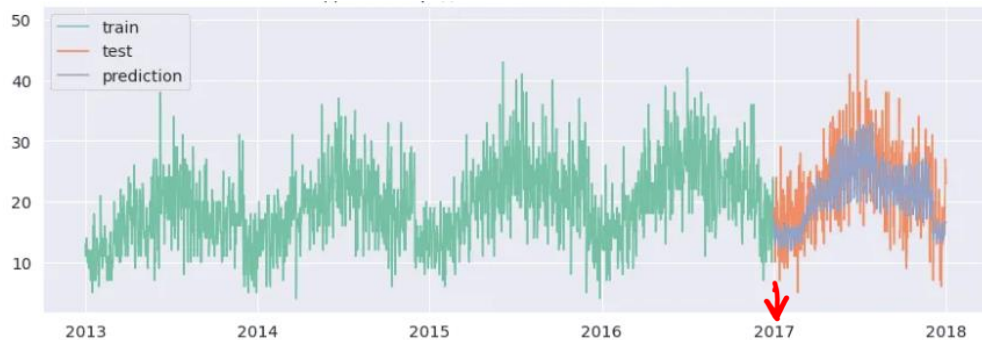
- I. INTRODUCTION
- II. TEMPORAL AND SPATIAL DATA ANALYSIS
- III. CONCLUSION



Things that we want to know about time series

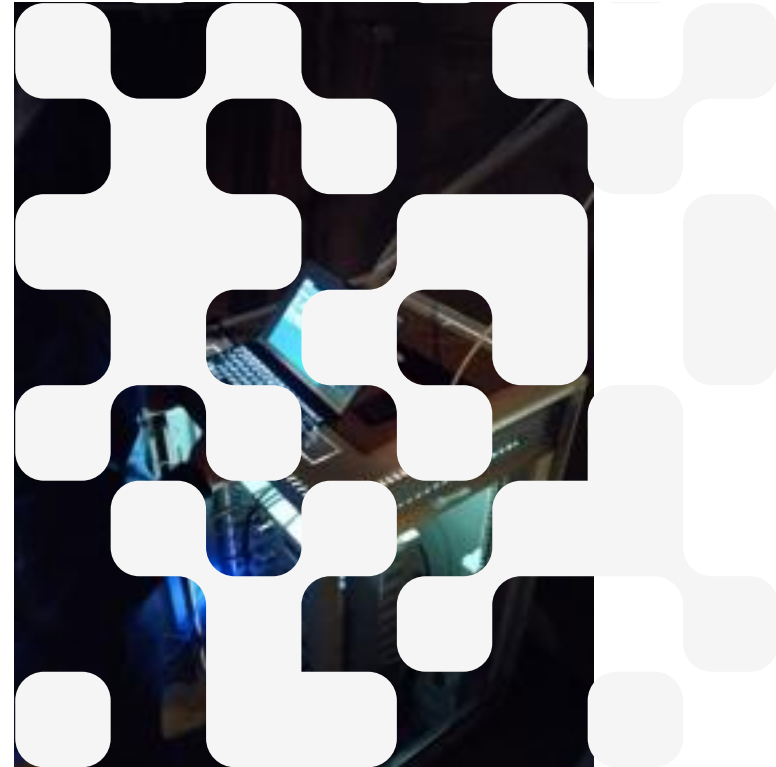
1. How is it going?
2. What will happen next step (few steps)?
3. When SMTH will happen?

→ Fault diagnosis
→ Prognostic modeling
→ RUL



Agenda

- I. INTRODUCTION
- II. TEMPORAL AND SPATIAL DATA ANALYSIS
- III. CONCLUSION



Hands on session: first of all, data filtering

Gaussian Blurring

In this method, instead of a box filter, a Gaussian kernel is used. It is done with the function, `cv.GaussianBlur()`. We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, `sigmaX` and `sigmaY` respectively. If only `sigmaX` is specified, `sigmaY` is taken as the same as `sigmaX`. If both are given as zeros, they are calculated from the kernel size. Gaussian blurring is highly effective in removing Gaussian noise from an image. The above code can be modified for Gaussian blurring:

```
blur = cv.GaussianBlur(img,(5,5),0)
```

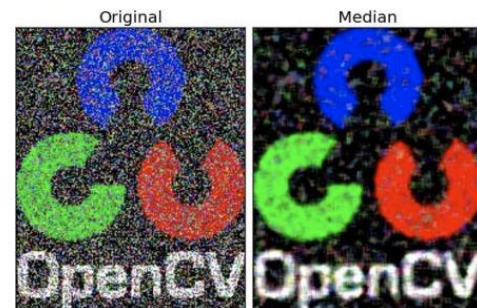


[Yandex Handbook on ML](#)

Median Blurring

Here, the function `cv.medianBlur()` takes the median of all the pixels under the kernel area and the central element is replaced with this median value. This is highly effective against salt-and-pepper noise in an image. Interestingly, in the above filters, the central element is a newly calculated value which may be a pixel value in the image or a new value. But in median blurring, the central element is always replaced by some pixel value in the image. It reduces the noise effectively. Its kernel size should be a positive odd integer. Check the result:

```
median = cv.medianBlur(img,5)
```



Hands on session: first of all, data filtering

1. Implement the code, compare the results obtained with the autoencoder, Gaussian filter, Fourier transform, Median filter
2. Enhance the autoencoder: use the noisy images as inputs and original (noiseless) images as targets.

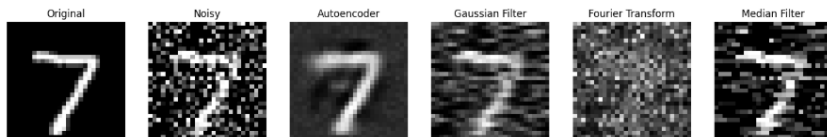
```
# Training the Autoencoder
for epoch in range(num_epochs):
    for data in train_loader:
        img, _ = data
        img = img.view(img.size(0), -1).to(device)
        noise = torch.randn(img.size()).to(device) * noise_factor
        noisy_img = img + noise
        noisy_img = torch.clamp(noisy_img, 0., 1.)

        # Forward pass
        output = model(noisy_img)
        loss = criterion(output, noisy_img)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

~~noisy~~ img



?

Fault diagnosis intuition

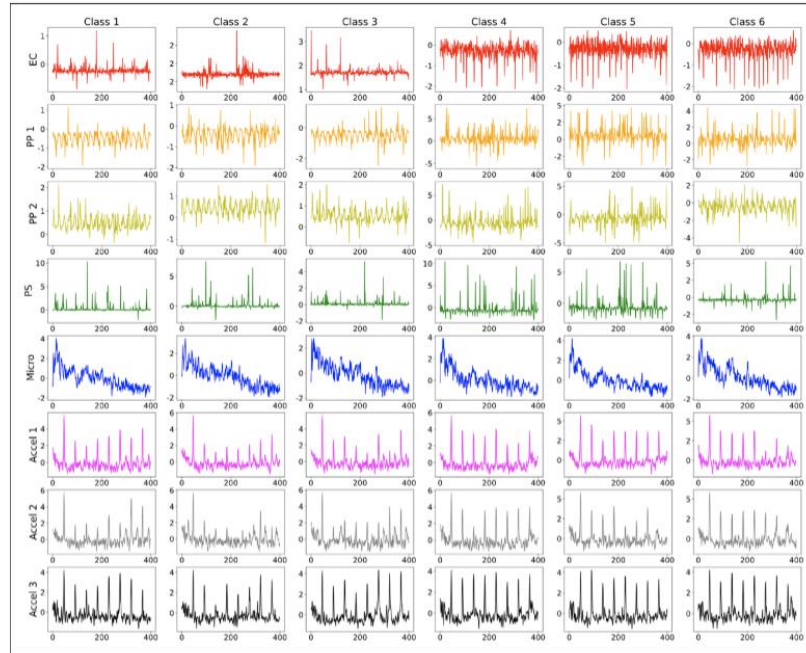
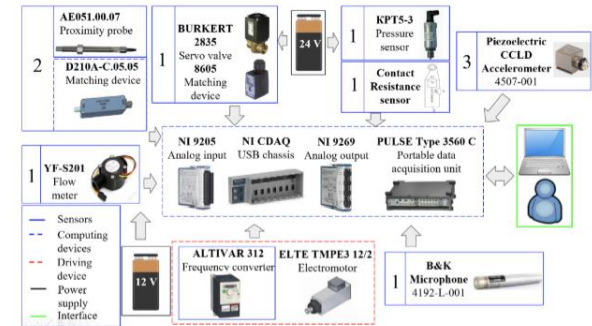
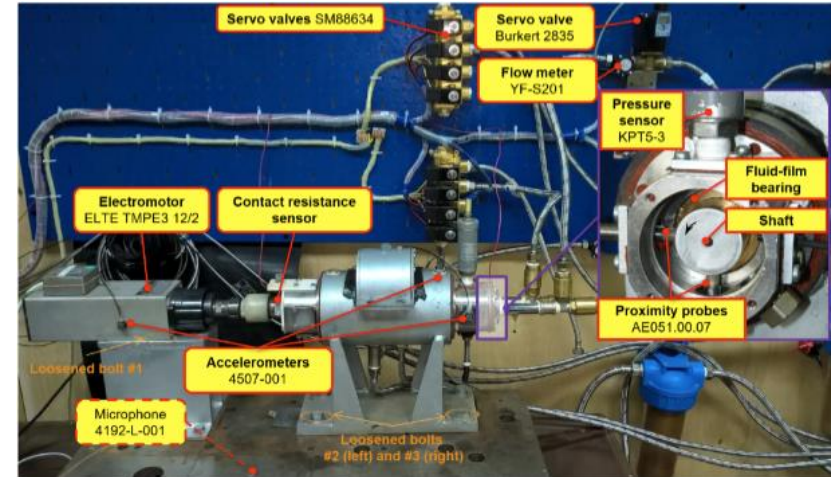


Figure 3. The measurement results in a form of training samples for all the classes of the test rig state: time series for the data obtained from the national instruments chassis (EC is electromotor current, PP 1, 2 are proximity probes, PS is pressure sensor) and the FFTs for the data obtained from the bruel&kjaer unit (micro is microphone, accel 1–3 are accelerometers). Description for sensors is given in Table 1.



[Fault diagnosis systems for rotating machines operating with fluid-film bearings, 2022](#)

Fault diagnosis intuition

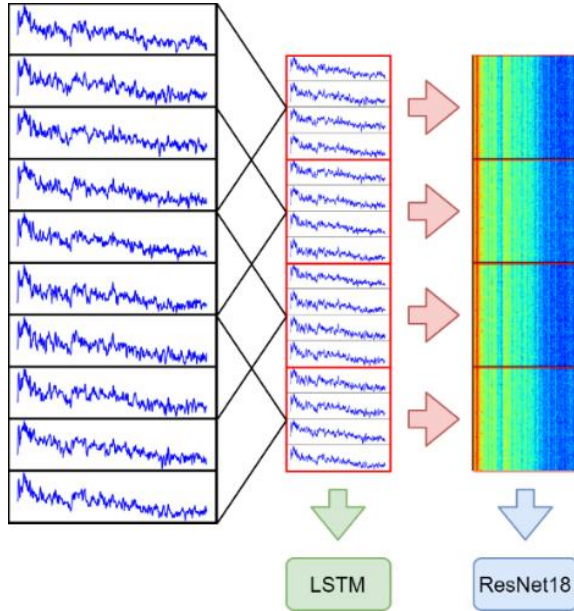
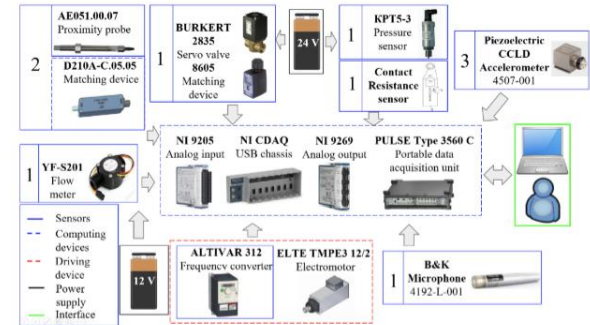
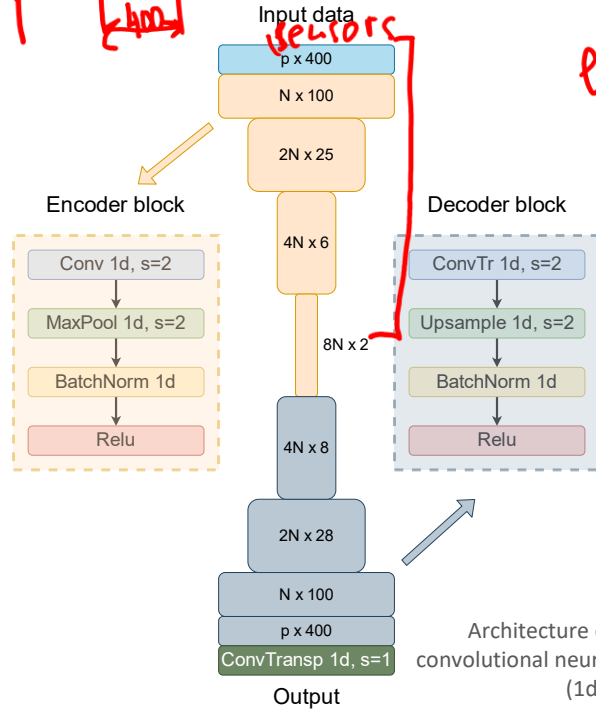


Table 3. The 6 classes classification results obtained using MLP and data from one of the sensors.

Sensor abbreviation	Train accuracy	Validation accuracy	Test accuracy
EC	0.478	0.324	0.373
PP 1	0.37	0.375	0.359
PP 2	0.485	0.509	0.479
PS	0.492	0.403	0.544
Micro	0.929	0.458	0.719
Accel 1	0.914	0.496	0.685
Accel 2	0.929	0.567	0.699
Accel 3	0.96	0.564	0.597

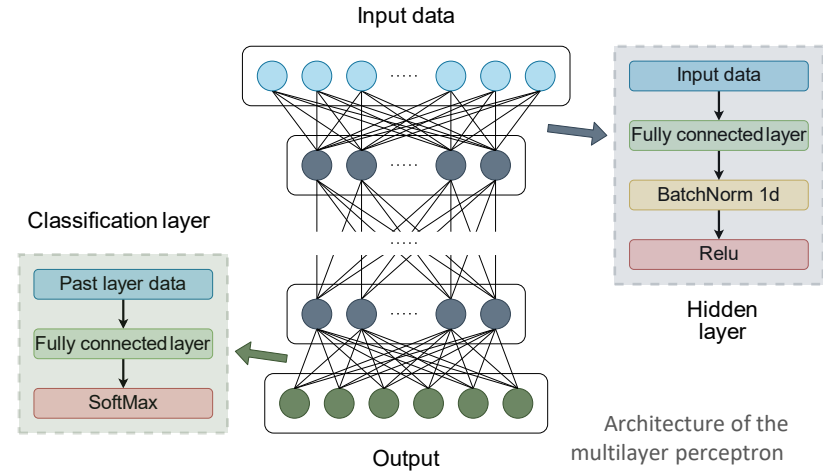


Fault diagnosis intuition



Net	Use pretrained encoder	Training accuracy	Validation accuracy	Test accuracy
MLP	x	0.987	0.790	0.801
1d CNN+MLP	-	0.991	0.767	0.806
1d CNN+LSTM+MLP	-	0.996	0.801	0.833
ResNet18	-	1.0	0.795	0.801
1d CNN+MLP	+	0.936	0.846	0.8
1d CNN+LSTM+MLP	+	0.997	0.811	0.82
ResNet18	+	1.0	0.782	0.804

Handwritten red note: "Error" with an arrow pointing to the 'Test accuracy' column.



Fault diagnosis in some good papers

SPRINGER NATURE Link


Find a journal Publish with us Track your research Search



Home > Circuits, Systems, and Signal Processing > Article

Transformers in Time-Series Analysis: A Tutorial

Published: 25 July 2023
Volume 42, pages 7433–7466, (2023) [Cite this article](#)

[Download PDF](#) Access provided by Innopolis University

Sabeen Ahmed , Ian E. Nielsen, Aakash Tripathi, Shamoon Siddiqui, Ravi P. Ramachandran & Ghulam Rasool

 10k Accesses  56 Citations  5 Altmetric [Explore all metrics](#)

Abstract

Transformer architectures have widespread applications, particularly in Natural Language Processing and Computer Vision. Recently, Transformers have been employed in various aspects of time-series analysis. This tutorial provides an overview of the Transformer architecture, its applications, and a collection of examples from recent research in time-series analysis. We delve into an explanation of the core components of the Transformer, including the self-attention mechanism, positional encoding, multi-head, and encoder/decoder. Several enhancements to the initial Transformer architecture are highlighted to tackle time-series tasks. The tutorial also provides best practices and techniques to overcome the challenge of effectively training Transformers for time-series analysis.

1-D Convolutional Neural Networks for Signal Processing Applications

Serkan Kiranyaz², Turker Ince², Osama Abdeljaber¹, Onur Avci¹, and Moncef Gabbouj¹

¹Department of Electrical Engineering, Qatar University, Qatar.

Email: s.kiranyaz@qu.edu.qa

²Electrical & Electronics Engineering Department, Izmir University of Economics, Turkey.

Email: turker.ince@izmirconet.edu.tr

³Department of Civil Engineering, Qatar University, Qatar.

Emails: a.abdeljaber@qu.edu.qa and onur.avci@qu.edu.qa

⁴Department of Signal Processing, Tampere University of Technology, Finland.

Email: Moncef.gabbouj@tut.fi

Abstract—1D Convolutional Neural Networks (CNNs) have recently become the *state-of-the-art* technique for crucial signal processing applications such as patient-specific ECG classification, structural health monitoring, anomaly detection in power electronics circuitry and motor-fault detection. This is an expected outcome as there are numerous advantages of using an adaptive and compact 1D CNN instead of a conventional (2D) deep counterparts. First of all, compact 1D CNNs can be efficiently trained with a limited dataset of 1D signals while the 2D deep CNNs, besides requiring 1D to 2D data transformation, usually need datasets with massive size, e.g., in the “Big Data” scale in order to prevent the well-known “overfitting” problem. 1D CNNs can directly be applied to the raw signal (e.g., current, voltage, vibration, etc.) without requiring any pre- or post-processing such as feature extraction, selection, dimension reduction, denoising, etc. Furthermore, due to the simple and compact configuration of such adaptive 1D CNNs that perform only linear 1D convolutions (scalar multiplications and additions), a real-time and low-cost hardware implementation is feasible. This paper reviews the major signal processing applications of compact 1D CNNs with a brief theoretical background. We will present their state-of-the-art performances and conclude with focusing on some major properties.

Keywords – 1-D CNNs, Biomedical Signal Processing, SHM

I. INTRODUCTION

Unlike the traditional Artificial Neural Networks (ANNs) Convolutional Neural Networks (CNNs) have the unique ability to fuse feature extraction and classification into a single learning body and thus eliminate the need for such fixed and hand-crafted features. Conventional deep CNNs (i.e. 2D CNNs) have been originally introduced to perform object recognition tasks for 2D signals such as images or video frames. They have recently become the *de-facto* standard for many Computer Vision and Pattern Recognition tasks within large data archives as they achieved the state-of-the-art performances [1]–[3]. However, the utilization of a conventional deep CNN for a 1-D signal processing application naturally requires a proper 1D to 2D conversion. For instance, recently, several researchers have attempted to use deep CNNs for fault diagnosis of bearings [4]–[11]. For this purpose, they have used different conversion techniques to represent the 1D vibration signals in 2D. The most commonly used technique is to directly reshape the vibration signal into an $n \times m$ matrix called “the vibration image” [9]. Another technique was used in [5] where two vibration signals were measured using two accelerometers. After that, Discrete

Fourier Transform (DFT) was applied, and then the two transformed signals were concatenated into a matrix which can be fed to a conventional deep CNN. For electrocardiogram (ECG) classification and arrhythmia detection, the common approach is to use power- or log-spectrogram to convert each ECG beat to a 2D image [12], [13]. However, there are certain drawbacks and limitations of using such deep CNNs. To start with it is known that they pose a high computational complexity that requires special hardware especially for training. Hence, 2D CNNs are not suitable for real-time applications on mobile and low-power/low-memory devices. Moreover, proper training of deep CNNs requires a very large training dataset in order to achieve a reasonable generalization capability. This may not be a viable option for many practical 1D signal applications where the labeled data is scarce.

To address these drawbacks, compact 1D CNNs have been recently developed to operate directly and more efficiently on 1D signals. They have achieved the state-of-the-art performance levels in several signal processing applications such as early arrhythmia detection in electrocardiogram (ECG) beats [15]–[17], on-site structural health monitoring [14], [18]–[22], motor fault detection [23] and real-time monitoring of high-power circuitry [24]. Additionally, two recent studies have utilized 1D CNNs for damage detection in bearings [25], [26]. However, in the latter study conducted by Zhang et al. [26], both single and ensemble of deep 1D CNNs were created to detect, localize, and quantify bearing faults. The deep configuration of 1D CNN used in this study consisted of 6 large convolutional layers followed by two fully connected layers. Other deep 1D CNN approaches have been recently followed by [27]–[29] for anomaly detection in ECG signals. These deep configurations share the common drawbacks of their 2D counterparts. For example, in [26], several “tricks” were used to improve the generalization performance of the deep CNN such as data augmentation, batch normalization, dropout, majority voting, etc. Another approach to tackle this problem is to use the majority of the dataset for training which may not be feasible at all in many practical applications. In the study [26] more than 96% of the total data is used to train the deep network. Hence the assumption that such a large set of training data will be available may hinder the usage of this method in practice. Therefore, in this study we shall draw the focus particularly on compact 1D CNNs with few hidden layers/neurons, and their applications to some major signal processing applications with the assumption that the labeled data is scarce and personalized or device-specific solutions are required to maximize the detection/identification accuracy.

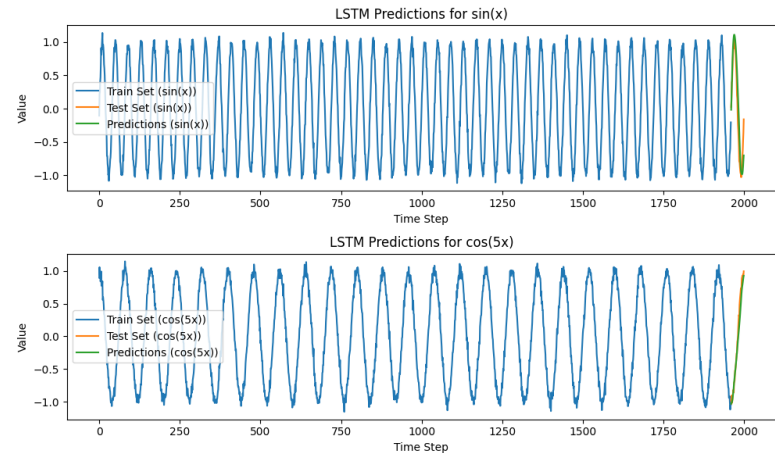
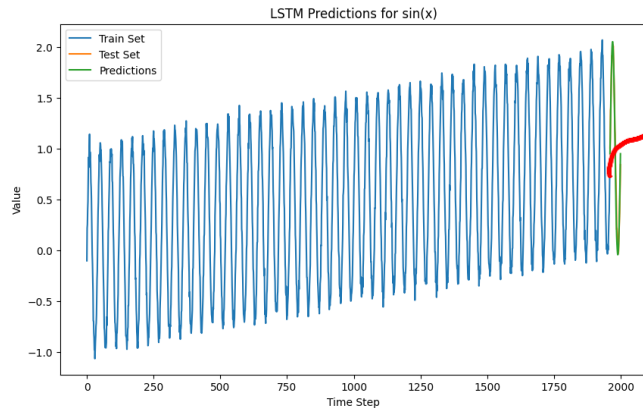
[Transformers in Time-Series Analysis: A Tutorial](#)

[1-D Convolutional NN for signal processing applications](#)

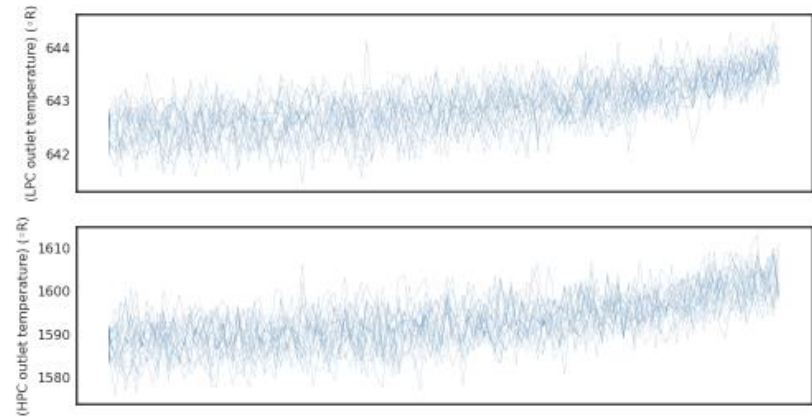
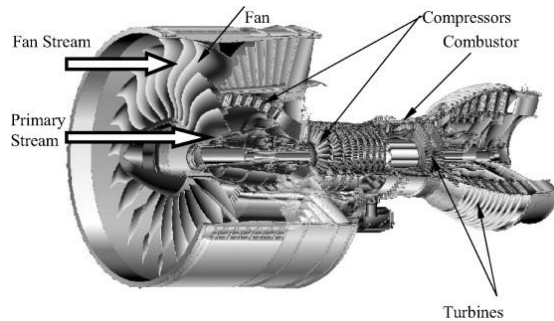
Predictive modeling

```
# LSTM model class
class myLSTM(nn.Module):
    def __init__(self, input_size=1, hidden_size=50, out_size=1):
        super().__init__()
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size)
        self.linear = nn.Linear(hidden_size, out_size)
        self.hidden = (torch.zeros(1,1,hidden_size), torch.zeros(1,1,hidden_size))

    def forward(self, seq):
        lstm_out, self.hidden = self.lstm(seq.view(len(seq), 1, -1), self.hidden)
        pred = self.linear(lstm_out.view(len(seq), -1))
        return pred[-1]
```



Remaining useful life estimation



In [13]: `df_train.head()`

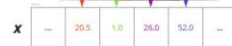
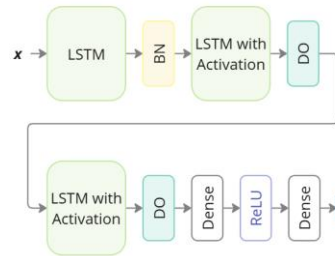
Out[13]:

	engine	cycle	setting_1	setting_2	(LPC outlet temperature) (°R)	(HPC outlet temperature) (°R)	(LPT outlet temperature) (°R)	(bypass-duct pressure) (psia)	(HPC outlet pressure) (psia)	(Physical fan speed) (rpm)	(Physical core speed) (rpm)
0	1	1	-0.0007	-0.0004	641.82	1589.70	1400.60	21.61	554.36	2388.06	9046.19
1	1	2	0.0019	-0.0003	642.15	1591.82	1403.14	21.61	553.75	2388.04	9044.07
2	1	3	-0.0043	0.0003	642.35	1587.99	1404.20	21.61	554.26	2388.08	9052.94
3	1	4	0.0007	0.0000	642.35	1582.79	1401.87	21.61	554.45	2388.11	9049.48
4	1	5	-0.0019	-0.0002	642.37	1582.85	1406.22	21.61	554.00	2388.06	9055.15

Remaining useful life estimation

Homoscedastic regression

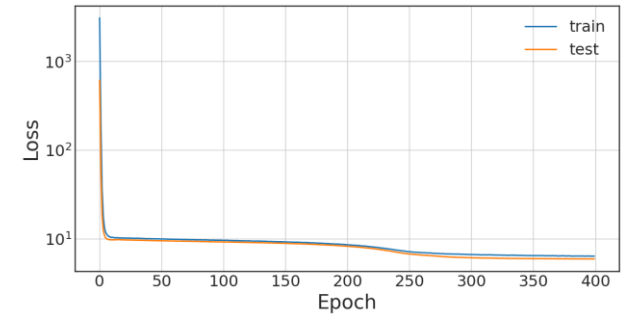
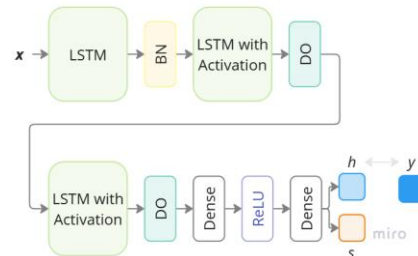
$$L_{MSE} = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - h^{(i)})^2$$



Activation (baseline):
 $a = \tanh()$
Activation (proposed):
 $a = \sin() + \cos()$

Heteroscedastic regression

$$L_{UA} = \frac{1}{m} \sum_{i=1}^m \left(\exp(-s^{(i)}) (y^{(i)} - h^{(i)})^2 + s^{(i)} \right)$$



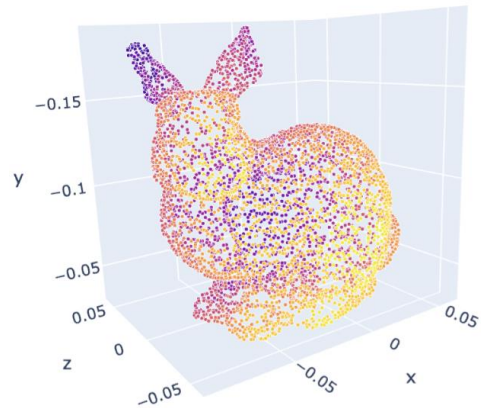
```
#Activation function by defaults is 'tanh'
#Proposed activation function
def sin0cos(x):
    return tf.sin(x) + tf.cos(x)

#Baseline LSTM model
def create_lstm_model(do=0.0, af=sin0cos):
    model = Sequential()
    model.add(LSTM(100, return_sequences=True, input_shape=(1, X_train.shape[1])))
    model.add(BatchNormalization())
    model.add(LSTM(50, return_sequences=True, activation=af))
    model.add(Dropout(do))
    model.add(LSTM(10, return_sequences=True, activation=af))
    model.add(Dropout(do))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(1))
    return model
```

	R2	RMSE	time to train	time to predict	total time
kNN	79.79%	18.68	0.032	0.032	0.064
SVM	78.13%	19.43	27.877	0.129	28.006
Random Forest	80.69%	18.26	7.647	0.203	7.850
LSTM+MSE	76.78%	20.03	1424.222	12.441	1436.663
LSTM+UANLL	81.68%	17.79	1417.211	21.575	1438.786

Uncertainty quantification in RUL project, 2024

Cloud of points



Cloud of points processing: PointNet

1. Point Feature Extraction

Each point p_i in the point cloud is represented as a vector $p_i = (x_i, y_i, z_i)$. The local features are extracted using a series of multi-layer perceptrons (MLPs):

$$f_i = \text{MLP}(p_i)$$

where f_i is the local feature vector for point p_i .

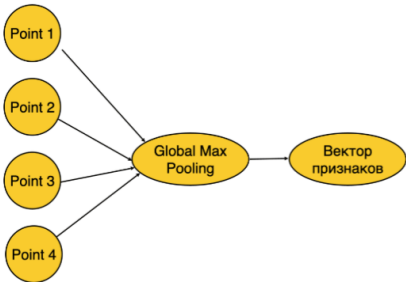
2. Transformation Network (T-Net)

The T-Net learns a transformation matrix T that aligns the point cloud to a canonical space:

$$T = \text{T-Net}(P)$$

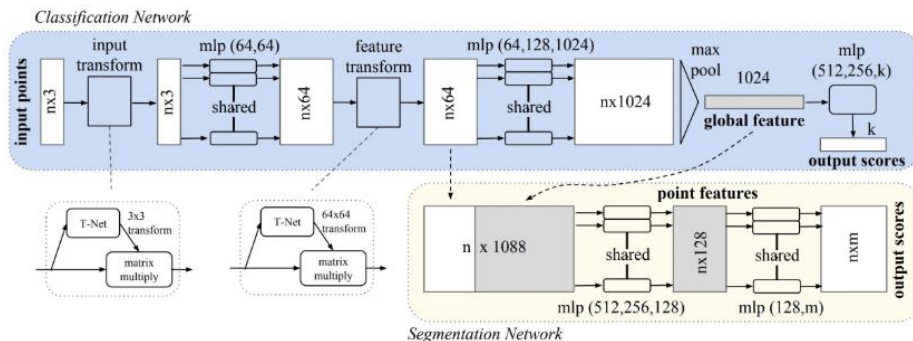
where P is the point cloud matrix (each row is a point vector). The transformed point cloud P' is:

$$P' = P \cdot T$$



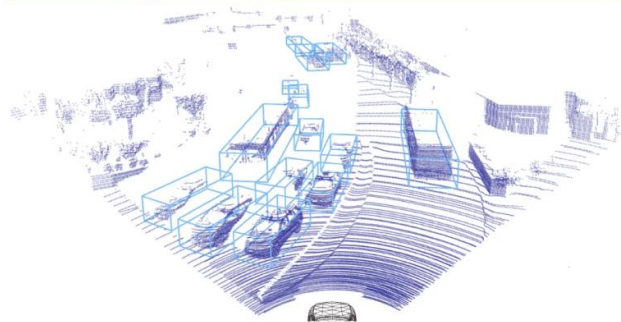
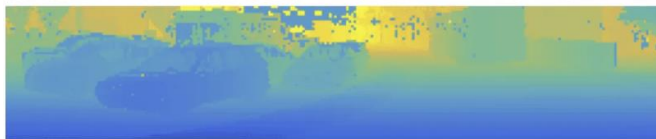
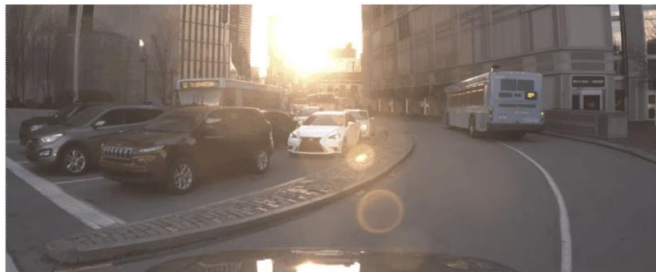
Component	Description
Input	Raw 3D point cloud data (unordered set of points).
Point Feature Extraction	Each point is processed individually to extract local features.
Transformation Network	Learns to align the point cloud to a canonical space using a series of T-Net (Transformation Network) layers.
Symmetric Function (Max Pooling)	Aggregates local features across all points to capture global information. This ensures permutation invariance.
Global Feature Vector	The output of the max pooling operation, representing the global shape information.
Classification Network	Uses the global feature vector to classify the entire point cloud into different categories.
Segmentation Network	Combines local and global features to predict per-point labels for segmentation tasks.
Loss Functions	- Classification Loss : Cross-entropy loss for classification tasks. - Segmentation Loss : Combination of cross-entropy loss and regularization terms for segmentation tasks.
Training	The network is trained using backpropagation and stochastic gradient descent (SGD) with appropriate loss functions.
Inference	Once trained, the network can process new point clouds to perform classification or segmentation tasks.

Cloud of points processing: PointNet



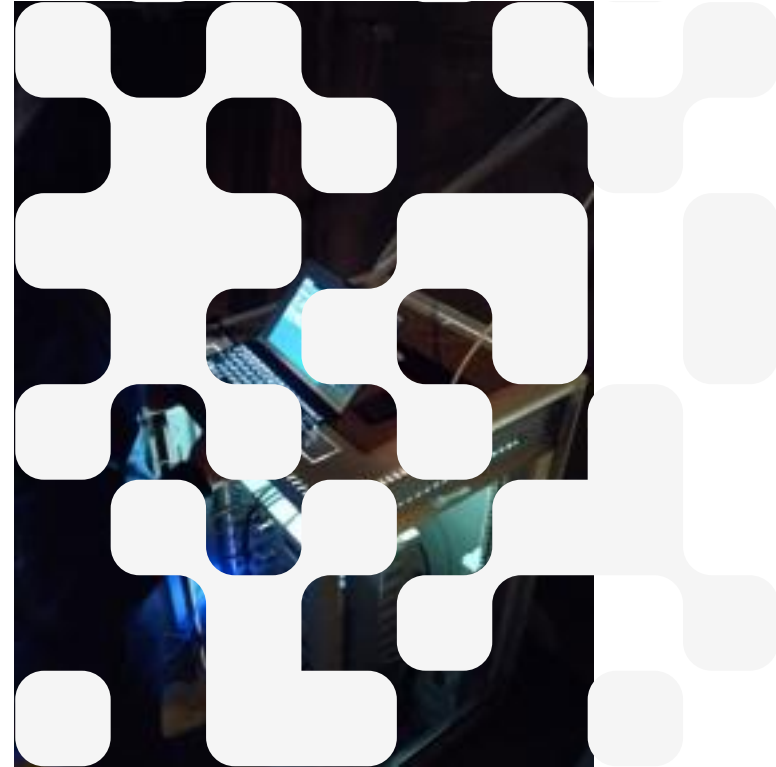
Component	Description
Input	Raw 3D point cloud data (unordered set of points).
Point Feature Extraction	Each point is processed individually to extract local features.
Transformation Network	Learns to align the point cloud to a canonical space using a series of T-Net (Transformation Network) layers.
Symmetric Function (Max Pooling)	Aggregates local features across all points to capture global information. This ensures permutation invariance.
Global Feature Vector	The output of the max pooling operation, representing the global shape information.
Classification Network	Uses the global feature vector to classify the entire point cloud into different categories.
Segmentation Network	Combines local and global features to predict per-point labels for segmentation tasks.
Loss Functions	<p>- Classification Loss: Cross-entropy loss for classification tasks.</p> <p>- Segmentation Loss: Combination of cross-entropy loss and regularization terms for segmentation tasks.</p>
Training	The network is trained using backpropagation and stochastic gradient descent (SGD) with appropriate loss functions.
Inference	Once trained, the network can process new point clouds to perform classification or segmentation tasks.

Plan B. Cylindrical projection



Agenda

- I. INTRODUCTION
- II. TEMPORAL AND SPATIAL DATA ANALYSIS
- III. CONCLUSION



Using AI in temporal and spatial data analysis is particularly cool for several reasons

1. Pattern Recognition and Anomaly Detection:

- **Temporal Data:** AI models, especially deep learning techniques like RNNs and LSTMs, excel at recognizing patterns and detecting anomalies in time-series data. This is crucial for applications like predictive maintenance, fraud detection, and stock market analysis.

- **Spatial Data:** AI can identify complex spatial patterns and relationships, which are often difficult for traditional methods to capture. This is useful in fields like geographic information systems (GIS), urban planning, and environmental monitoring.

2. Scalability and Efficiency:

- AI algorithms can handle large volumes of temporal and spatial data efficiently. Techniques like convolutional neural networks (CNNs) and graph neural networks (GNNs) are designed to process and analyze vast datasets, making them ideal for big data applications.

3. Adaptability and Learning:

- AI models can learn and adapt to new data over time, improving their accuracy and performance. This adaptability is particularly valuable in dynamic environments where data characteristics may change frequently.

4. Integration of Multiple Data Sources:

- AI can integrate and analyze data from multiple sources, including temporal and spatial data. This holistic approach allows for more comprehensive and accurate insights, which is beneficial in interdisciplinary fields like climate science, epidemiology, and transportation planning.

5. Predictive Analytics:

- AI models can make accurate predictions based on historical and real-time data. For example, in traffic management, AI can predict congestion patterns and suggest

optimal routes. In healthcare, AI can predict disease outbreaks based on temporal and spatial data.

6. Automated Decision-Making:

- AI can automate decision-making processes by analyzing temporal and spatial data in real-time. This is particularly useful in applications like autonomous vehicles, smart grids, and emergency response systems.

7. Enhanced Visualization:

- AI can enhance data visualization techniques, making it easier to interpret complex temporal and spatial data. Tools like generative adversarial networks (GANs) can create realistic simulations and visualizations, aiding in better understanding and decision-making.

8. Personalization and Customization:

- AI can provide personalized and customized insights based on individual preferences and behaviors. For example, in retail, AI can analyze shopping patterns over time and across different locations to offer personalized recommendations.

9. Complex Relationship Modeling:

- AI can model and understand complex relationships between temporal and spatial variables, which are often non-linear and interdependent. This capability is crucial in fields like economics, ecology, and social sciences.

10. Innovative Applications:

- AI opens up new possibilities for innovative applications in various domains. For instance, AI-driven drones can analyze spatial data for agricultural monitoring, while AI-powered wearables can track temporal health data for personalized healthcare.

And what do you expect most from AI in the near future?

And one more time. Why AI?

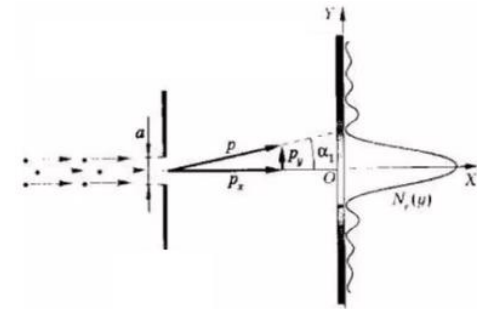
Initial conditions		+	Set of the equations		+	Boundary conditions	
Equation / law	in tensor form		in scalar form (in Cartesian coordinates)			Cumulative sum of...	
						unknown values	equations
Equation of motion	$\nabla \cdot T_{\sigma} + \rho \vec{f} = \rho \frac{d\vec{v}}{dt}$		$\frac{\partial \sigma_{ij}}{\partial x_j} + \rho f_i = \rho \frac{dv_i}{dt}$			10 σ_{ij}, v_i	3
Newton's law	$D_{\sigma} = 2\mu D_{\xi}, \quad S_{\sigma} = (3\lambda + 2\mu)S_{\xi}$		$\left(\sigma_{ij} - \delta_{ij} \frac{\sigma_{mm}}{3} \right) = 2\mu \left(\xi_{ij} - \delta_{ij} \frac{\xi_{kk}}{3} \right), \quad \sigma_{mm} = (3\lambda + 2\mu)\xi_{kk}$			18 ξ_{ij}, μ, λ	9
Continuity equation	$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0$		$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho v_i)}{\partial x_i} = 0$			18	10
Stokes formula	$T_{\xi} = \frac{1}{2} (\nabla \otimes \vec{v} + \vec{v} \otimes \nabla)$		$\xi_{ij} = \frac{1}{2} \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right)$			18	16
Heat balance equation	$\frac{d\theta}{dt} = \frac{1}{C_p \rho} \nabla \cdot (\lambda \nabla \theta) + \frac{T_{\sigma} \cdot T_{\xi}}{C_p \rho}$		$\frac{\partial \theta}{\partial t} + \frac{\partial \theta}{\partial x_i} v_i = \frac{1}{C_p \rho} \frac{\partial}{\partial x_j} \left(\lambda \frac{\partial \theta}{\partial x_j} \right) + \frac{\sigma_{km} \xi_{km}}{C_p \rho}$			19 + θ	17
Rheology equation	-		$\mu = \mu(T_{\xi}, S_{\sigma}, \theta), \quad \lambda = \lambda(T_{\xi}, S_{\sigma}, \theta)$			19	19
						19	19



A hydromechanical system

Heisenberg uncertainty principle:

$$\Delta p_y \Delta y \geq h/(2\pi).$$



Kind of a generalized Heisenberg uncertainty principle in application to deterministic mathematical modeling:

the more accurate the physical correspondence of the model to the object, the greater its computational error

Thank you for your attention!

a.kornaev@innopolis.ru, [@avkornaev](https://twitter.com/avkornaev)

