

Project Report On

Signal Denoising Using Convolutional and LSTM Autoencoders on Synthetic Time-Series Data

By

Arijit Ghosh (35000121017)

Niloy Acharyya (35000121031)

Ratan Bhowmick (35000121072)

Sailesh Yadav (35000122041)

Mainak Dutta (35000122044)

A comprehensive project report has been submitted in partial fulfilment
of the requirements for the degree of

Bachelor of Technology in COMPUTERSCIENCE &ENGINEERING

Under the supervision

Of

Dr. PRASUN HALDER

Head of the Department of CSE Dept.

**RAMKRISHNA MAHATO GOVERNMENT
ENGINEERING COLLEGE**

AGHARPUR , RAMAMOTI , JOYPUR

PURULIA – 723103 , WEST BENGAL , INDIA

CERTIFICATE OF APPROVAL

This is to certify that Arijit Ghosh (35000121017), Niloy Acharyya (35000121031), Ratan Bhowmick (35000121072), Sailesh Yadav (35000122041) and Mainak Dutta (35000122044), successfully completed the project titled " Signal Denoising Using Convolutional and LSTM Autoencoders on Synthetic Time-Series Data" at Ramkrishna Mahato Government Engineering College under my supervision and guidance in the fulfilment of requirements of 8th Semester, Bachelor of Technology (Computer Science and Engineering) of Maulana Abul Kalam Azad University of Technology, Kolkata, West Bengal.

.....
(Signature)

Dr. Prasun Halder

Head of the Department

Department of Computer Science & Engineering

Ramkrishna Mahato Government Engineering College, Purulia

DECLARATION

“We do hereby declare that this progress report submission is our own work conformed to the norms and guidelines given in the Ethical Code of Conduct of the Organization and that, to the best of our knowledge and belief, it contains no material written earlier by another neither person nor material (data, theoretical analysis, figures, and text) which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text”

Arijit Ghosh (35000121017)

Niloy Acharyya (35000121031)

Ratan Bhowmick (35000121072)

Sailesh Yadav (35000122041)

Mainak Dutta (35000122044)

Date:

Place: PURULIA

.....

(Signature)

Dr. Prasun Halder

Head of the Department

Department of Computer Science & Engineering

Ramkrishna Mahato Government Engineering College, Purulia

ACKNOWLEDGEMENT

We deem it a pleasure to acknowledge our sense of gratitude to our project guide Dr. Prasun Halder under whom we have carried out the project work. His incisive and objective guidance and timely advice encouraged us with constant flow of energy to continue the work.

We wish to reciprocate in full measure the kindness shown by Dr. Prasun Halder (H.O.D. Computer Science and Engineering) who inspired us with his valuable suggestions in successfully completing the project work.

We shall remain grateful to Dr. Bibek Chakrabarti, Principal, Ramkrishna Mahato Government Engineering College, for providing us a strong academic atmosphere by enforcing strict discipline to do the project work with utmost concentration and dedication.

Finally, we must say that no height is ever achieved without some sacrifices made at some end and it is here where we owe our special debt to our parents and our friends for showing their generous love and care throughout the entire period of time.

.....
(Signature)

Arijit Ghosh (35000121017)

.....
(Signature)

Niloy Achrayya (35000121031)

.....
(Signature)

Ratan Bhowmick (35000121072)

.....
(Signature)

Sailesh Yadav (35000122041)

.....
(Signature)

Mainak Dutta (35000122044)

ABSTRACT

Noise in time-series signals presents a major challenge in domains like biomedical monitoring, where signal accuracy is crucial for analysis and diagnosis. This project addresses the problem of signal denoising using deep learning-based autoencoders. A synthetic sinusoidal dataset, designed to mimic biomedical signals, was corrupted with random noise to simulate real-world scenarios. Two architectures were implemented: a 1D Convolutional Autoencoder (Conv1D) and an LSTM Autoencoder. The models were trained on windowed and normalized data using the Mean Squared Error (MSE) loss function. Initial results showed basic learning, but limited denoising performance.

To improve results, an enhanced Conv1D autoencoder was developed, leading to a significant boost in performance. It achieved an MSE of 0.016, a Root Mean Squared Error (RMSE) of 0.127, and a Signal-to-Noise Ratio (SNR) improvement of 7.10 dB. The model was able to reconstruct clean signals effectively, as confirmed by both quantitative metrics and qualitative plots. This project demonstrates the power of deep learning for time-series denoising and lays the groundwork for extending such techniques to real-world biomedical signals like ECG and PPG in healthcare monitoring systems.

CONTENTS

1. Problem Description.....	07
2.Literature Survey.....	09
3. Introduction.....	17
4. Methodology	18
5. Experiment and Result.....	23
6. Conclusion	32
7. References.....	34

1. PROBLEM DESCRIPTION

1.1 Background of Signal Denoising

Signals, in their essence, are representations of physical phenomena, carrying vital information across diverse fields. From the intricate electrical impulses of a human heart captured by an Electrocardiogram (ECG) to the fluctuating radio waves conveying telecommunication data, and the subtle vibrations detected by IoT sensors, signals are fundamental to understanding and interacting with our environment. The integrity and purity of these signals are paramount, as their quality directly influences the accuracy of measurements, reliability of systems, and validity of subsequent analyses.

However, in real-world scenarios, signals are rarely pristine. They are invariably corrupted by **noise**, which can originate from a multitude of sources. This noise, often unwanted and random fluctuations, can be inherent to the measurement system (e.g., thermal noise in electronics), external environmental interferences (e.g., electromagnetic interference, acoustic background noise), or even human-induced artifacts. The impact of noise is profound and pervasive:

- **Medical Diagnostics:** In medical signals like ECGs, Electroencephalograms (EEGs), or Magnetic Resonance Imaging (MRI), noise can obscure crucial diagnostic features, leading to misinterpretations or delayed diagnoses. A noisy ECG could mask an arrhythmia, while a corrupted EEG might hide epileptic spikes.
- **Communication Systems:** In audio communication, background noise severely degrades speech intelligibility, making conversations difficult and reducing the efficiency of voice recognition systems. In wireless communication, noise can corrupt data packets, leading to retransmissions and reduced bandwidth.
- **Industrial and IoT Applications:** Sensors deployed in industrial environments or IoT networks often collect data contaminated by electrical interference, mechanical vibrations, or environmental fluctuations. This noisy data can lead to inaccurate readings, faulty control decisions, and unreliable system performance, impacting predictive maintenance or anomaly detection.
- **Scientific Research:** In fields like astrophysics, seismology, or materials science, subtle signals are often buried under significant noise, making their detection and analysis challenging and requiring advanced processing techniques to extract meaningful information.

The presence of noise directly diminishes the **Signal-to-Noise Ratio (SNR)**, making it difficult to discern the true underlying information. This degradation necessitates a crucial preprocessing step: **signal denoising**. The primary goal of denoising is to effectively separate the desired signal from the unwanted noise, thereby enhancing signal quality, improving feature extraction, and ensuring the reliability of subsequent processing, analysis, or decision-making.

1.2 Specific Problem Addressed

This project specifically addresses the challenge of **reducing noise from time-series amplitude data** using advanced machine learning techniques, particularly deep learning autoencoders. The dataset under consideration comprises time-series signals characterized by a clear, underlying sinusoidal pattern corrupted by additive noise, resulting in a "noised amplitude" series.

Traditional signal denoising methods, such as simple low-pass filtering (e.g., using Butterworth or FIR filters), moving averages, or basic Fourier Transform-based noise reduction, often operate under assumptions about the noise's spectral characteristics (e.g., distinct frequency bands for signal and noise) or rely on the noise being stationary. While effective for simple, well-behaved noise, these conventional approaches frequently encounter significant limitations when dealing with more complex scenarios:

- **Overlapping Spectra:** When the frequency content of the noise overlaps considerably with that of the desired signal, traditional filters can inadvertently remove valuable signal components along with the noise, leading to signal distortion or loss of critical information.
- **Non-Stationary Noise:** Many real-world noise sources are non-stationary, meaning their statistical properties (e.g., mean, variance, power spectrum) change over time. Fixed-parameter traditional filters struggle to adapt to such dynamic noise patterns.
- **Non-Linear Noise:** Noise that is not simply additive or Gaussian, or that interacts with the signal in complex, non-linear ways, is poorly handled by linear filtering techniques.
- **Lack of Prior Knowledge:** Traditional methods often require prior knowledge about the noise characteristics (e.g., noise power, specific frequency range) or the signal's properties, which may not always be available or easy to estimate in dynamic environments.
- **Manual Tuning:** Many classical filtering techniques require manual tuning of parameters (e.g., cutoff frequencies, filter order), a process

that can be time-consuming and suboptimal for varying signal conditions.

Given these limitations, there is a compelling need for more adaptive and robust denoising techniques. Deep learning models, with their ability to learn intricate, non-linear patterns directly from data without explicit feature engineering or rigid assumptions about noise distribution, offer a promising solution. This project leverages the power of deep autoencoders to learn an optimal mapping from noisy input signals to their corresponding clean versions, thereby circumventing the constraints typically faced by conventional methods. The core problem, therefore, is to effectively train deep neural networks that can discern and isolate the underlying signal structure from complex noise contamination in time-series amplitude data.

2. LITERATURE SURVEY

The field of signal processing has seen continuous evolution in techniques for mitigating noise, adapting from classical statistical methods to modern machine learning paradigms. This chapter provides a comprehensive review of relevant literature, starting with traditional signal denoising approaches, transitioning to the fundamentals of deep learning for sequence data, and then detailing the specific deep learning architectures (Autoencoders, Conv1D, and LSTM) foundational to this project. Finally, it outlines the scope for a review of related works in the domain.

2.1 Traditional Signal Denoising Techniques

Traditional signal denoising methods often rely on statistical assumptions about the signal and noise, or operate within specific domains (time, frequency, or time-frequency). While effective for certain types of noise, they frequently face limitations when dealing with complex, non-stationary, or non-linear noise characteristics.

- **Moving Average (MA) Filters:**
 - **Basic Principle:** This is one of the simplest and most widely used digital filters. It operates by calculating the average of a fixed number of data points (window) around each point in the time series. This process effectively smooths out rapid fluctuations by replacing each data point with the average of itself and its neighbors.

- **Strengths:** Easy to implement, computationally inexpensive, and effective at reducing random, high-frequency noise.
- **Limitations:** Introduces a phase delay (lag), can blur sharp features or sudden changes (e.g., peaks, edges) in the signal, and is not effective for complex or low-frequency noise that overlaps with the signal's bandwidth. The choice of window size is critical and often empirical.
- **Gaussian Filters:**
 - **Basic Principle:** Similar to the moving average, but it uses a weighted average where the weights are determined by a Gaussian function. Points closer to the center of the window are given higher weights, resulting in a smoother transition and less abrupt smoothing than a simple moving average.
 - **Strengths:** Provides excellent smoothing while preserving more of the signal's shape compared to a simple moving average. Particularly effective for noise that follows a Gaussian distribution.
 - **Limitations:** Can still blur signal features. Requires tuning of the standard deviation (sigma) of the Gaussian kernel, which dictates the degree of smoothing. Like other linear filters, its effectiveness diminishes when noise characteristics are non-Gaussian or highly complex.
- **Fourier Transform-based Filtering:**
 - **Basic Principle:** This method transforms the time-domain signal into the frequency domain using the Fast Fourier Transform (FFT). In the frequency domain, noise components (especially high-frequency or specific harmonic noise) can often be distinguished and isolated from the signal's dominant frequencies. Denoising is achieved by selectively attenuating or removing frequency components identified as noise, followed by an Inverse FFT (IFFT) to reconstruct the signal in the time domain.
 - **Strengths:** Highly effective when the signal and noise occupy distinct frequency bands. Powerful for removing periodic noise.
 - **Limitations:** Assumes stationarity of the signal and noise (i.e., their frequency content does not change significantly over time). Loss of temporal localization: modifications in the frequency domain affect the entire signal, potentially smearing out transient features in the

time domain. It struggles when signal and noise spectra significantly overlap.

- **Wavelet Denoising:**

- **Basic Principle:** Wavelet transforms decompose a signal into different frequency components across various time scales, providing a localized representation in both time and frequency domains. Denoising is performed by applying a threshold to the wavelet coefficients. Small coefficients, typically associated with noise, are set to zero or shrunk, while larger coefficients, representing significant signal features, are retained. An inverse wavelet transform reconstructs the denoised signal.
- **Strengths:** Superior to Fourier methods for non-stationary signals and signals containing transient features (e.g., spikes, abrupt changes), as it preserves local time-frequency information. Offers excellent energy compaction for many signals.
- **Limitations:** The choice of wavelet basis function (e.g., Haar, Daubechies) and the thresholding method (e.g., hard, soft, universal) can significantly impact performance and are often empirically determined. Can introduce artifacts if thresholding is too aggressive or inappropriate.

2.2 Fundamentals of Deep Learning for Sequence Data

Deep learning, a subset of machine learning, has revolutionized various fields by enabling computational models to learn hierarchical representations of data with multiple processing layers. Unlike traditional methods that often rely on hand-crafted features, deep neural networks (DNNs) can automatically extract intricate patterns and features directly from raw input.

- **Introduction to Neural Networks:** At their core, neural networks consist of interconnected layers of "neurons" or nodes. Each connection has a weight, and each neuron has a bias. Data flows through these layers, undergoing non-linear transformations via activation functions (e.g., ReLU, Sigmoid, Tanh). This layered structure allows DNNs to model highly complex and non-linear relationships within the data, making them powerful tools for tasks like pattern recognition, classification, and regression. Their ability to learn high-level abstractions from raw data circumvents the need for explicit feature engineering, a labor-intensive and often suboptimal process in traditional signal processing.

- **Suitability for Time-Series Analysis:** Deep learning, particularly with specialized architectures like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), is exceptionally well-suited for analyzing time-series data due to several inherent capabilities:
 - **Capturing Temporal Dependencies:** Time-series data inherently possesses sequential dependencies where the current observation is influenced by past observations. RNNs (and their variants like LSTMs) are specifically designed with internal memory mechanisms to capture and learn these long-range temporal correlations. CNNs, especially 1D convolutions, can also capture local temporal patterns through their receptive fields.
 - **Automatic Feature Extraction:** Instead of relying on predefined features (e.g., statistical moments, frequency components), deep learning models can automatically learn optimal features directly from the raw time-series data relevant to the task (e.g., noise characteristics, signal shapes).
 - **Handling Non-Linear Relationships:** Real-world signals and noise often exhibit complex, non-linear interactions. Deep learning's inherent non-linearity, introduced by activation functions and multiple layers, allows it to model these complex relationships far more effectively than linear traditional filters.
 - **Scalability:** With increasing data availability, deep learning models can scale effectively, leveraging larger datasets to learn more robust and generalizable representations of signals and noise.

2.3 Autoencoders (AE) for Denoising

Autoencoders are a class of artificial neural networks designed for unsupervised learning of efficient data encodings. Their primary objective is to learn a compressed representation (encoding) of the input data and then reconstruct the original input from this compressed representation as accurately as possible.

- An autoencoder typically consists of two main parts:
 - **Encoder:** This part maps the high-dimensional input data (x) to a lower-dimensional latent-space representation (z). The encoder function can be denoted as $z = f(x)$.
 - **Decoder:** This part maps the latent-space representation back to the original input space, aiming to reconstruct the input as accurately as possible ($x' = g(z)$).

- **Bottleneck:** The latent-space layer (also known as the "bottleneck" or "code" layer) is crucial. It is typically a layer with fewer units than the input and output layers, forcing the encoder to learn a compressed, yet highly informative, representation of the input. This compression compels the autoencoder to focus on the most salient features of the data, discarding redundancy.
- **Reconstruction Task:** The autoencoder is trained by minimizing a reconstruction loss function (e.g., Mean Squared Error - MSE) between the input x and its reconstruction x' . The goal is for x' to be as close to x as possible.
- **Adaptation for Denoising (Denoising Autoencoders - DAE):**
 - The standard autoencoder learns to reconstruct its *clean* input. Denoising autoencoders (DAEs) extend this concept by introducing noise to the input during training. Specifically, a DAE is trained to reconstruct the *original, clean* data from a *corrupted (noisy)* version of that data.
 - During training, the DAE is fed noisy input x_{noisy} , but its target output is the corresponding clean signal x_{clean} . This forces the encoder to learn robust features that capture the underlying signal structure and are invariant to the applied noise. The decoder then learns to map these robust features back to the clean signal space, effectively suppressing the noise introduced in the input. This makes DAEs powerful tools for learning intrinsic data representations that are resilient to corruption.

2.4 Convolutional Autoencoders (CAE) for Signal Processing

Convolutional Autoencoders (CAEs) integrate convolutional layers into the autoencoder architecture, making them particularly effective for data with grid-like topologies, such as images (2D convolutions) and time-series signals (1D convolutions). For time-series data, 1D convolutions are employed.

- **Focus on 1D Convolutions for Time-Series Data:**
 - In a 1D CNN, a **kernel (or filter)**, which is a small matrix of learnable weights, slides across the input time series. At each position, it performs a dot product with the segment of the input data it covers. This operation detects local patterns or features (e.g., specific frequencies, transient events, rising/falling edges) within the sequential data.

- By applying multiple kernels, the network can learn to detect a variety of different local patterns, forming a feature map.
- **Explanation of Key Concepts in CAE Architecture:**
 - **Filters/Kernels:** These are the primary learnable components of a convolutional layer. Each filter is designed to activate when it detects a specific feature in its receptive field. In 1D, a filter moves along the time axis.
 - **Kernel Size:** This parameter defines the width of the filter, i.e., how many time steps the filter considers at once. A larger kernel size allows the filter to capture broader local temporal dependencies.
 - **Activation Functions:** Non-linear functions (e.g., Rectified Linear Unit - ReLU) applied after each convolutional operation. They introduce non-linearity into the model, enabling it to learn complex mappings that linear models cannot.
 - **Padding:** Determines how the input sequence is padded (usually with zeros) before convolution. 'Same' padding ensures that the output sequence length is the same as the input length, which is beneficial in autoencoders where reconstruction of the original length is desired. 'Valid' padding results in a smaller output.
 - **Pooling (MaxPooling1D):** Typically used in the encoder section, pooling layers reduce the dimensionality of the feature maps. MaxPooling1D takes the maximum value from a fixed-size window, effectively downsampling the feature map while retaining the most prominent features. This helps to make the model more robust to small shifts in the input signal and reduces computational complexity.
 - **Upsampling (UpSampling1D):** Employed in the decoder section, upsampling layers perform the inverse operation of pooling. UpSampling1D increases the dimension of the feature maps by repeating values or inserting zeros. This is crucial for gradually expanding the compressed representation back to the original signal's length and detail during the reconstruction phase.
- **CAE for Denoising:** The encoder part of a CAE uses convolutional and pooling layers to progressively extract hierarchical, compressed features from the noisy input signal. These features represent the underlying clean signal. The decoder then uses upsampling and convolutional layers to

reconstruct the clean signal from this compressed, noise-invariant representation. The convolutional nature allows the CAE to effectively learn local spatial/temporal correlations crucial for distinguishing signal from noise.

2.5 Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) Networks

Recurrent Neural Networks (RNNs) are a class of neural networks specifically designed to process sequential data by maintaining an internal "memory" that allows information to persist across time steps.

- **Brief Introduction to RNNs and Their Limitations:**

- **Basic Principle:** Unlike feedforward networks, RNNs have connections that loop back on themselves, allowing them to pass information from one step in a sequence to the next. This recurrent connection enables them to process sequences of arbitrary length and learn temporal dependencies.
- **Limitations (Vanishing/Exploding Gradients):** While conceptually powerful, vanilla RNNs suffer from practical limitations, most notably the vanishing gradient problem. During backpropagation through time (BPTT), gradients can become extremely small, making it difficult for the network to learn long-term dependencies (i.e., relationships between data points far apart in the sequence). Conversely, exploding gradients can also occur, leading to unstable training.

- **In-Depth Explanation of LSTM Cells:**

- To overcome the vanishing gradient problem, Long Short-Term Memory (LSTM) networks were introduced. LSTMs are a special kind of RNN that are capable of learning long-term dependencies. They achieve this through a sophisticated internal structure called an **LSTM cell**, which contains a **cell state** (C_t) and a **hidden state** (h_t), regulated by three specialized "gates":
 - **Cell State (C_t):** This acts as the "memory" of the network, carrying information through the sequence. It's like a conveyor belt, allowing information to flow along, with minor linear interactions.
 - **Hidden State (h_t):** This is the output of the LSTM cell at the current time step, also fed as input to the next time step.

- **The Three Gates (Sigmoid Activation):** Each gate uses a sigmoid activation function to output a value between 0 and 1, which represents how much of a particular piece of information should pass through.
 - **Forget Gate (f_t):** This gate decides what information from the previous cell state (C_{t-1}) should be thrown away. It takes the previous hidden state (h_{t-1}) and current input (x_t) as input.
 - **Input Gate (i_t):** This gate decides what new information should be stored in the current cell state. It has two parts: a sigmoid layer that decides which values to update, and a tanh layer that creates a vector of new candidate values (\tilde{C}_t) to be added to the state.
 - **Output Gate (o_t):** This gate decides what part of the cell state to output as the hidden state (h_t). It takes the previous hidden state (h_{t-1}) and current input (x_t) to decide which parts of the cell state (C_t) to filter through a tanh function.
- **How LSTMs Handle Long-Term Dependencies:** The gates allow LSTMs to selectively add, remove, or update information in the cell state. This fine-grained control over the flow of information enables LSTMs to learn dependencies spanning hundreds or even thousands of time steps, making them highly effective for tasks involving long sequences like speech recognition, natural language processing, and complex time-series analysis.
- **Application of LSTMs within an Autoencoder Framework:**
 - In an LSTM autoencoder for denoising, LSTMs are used in both the encoder and decoder components to leverage their sequential processing capabilities.
 - **Encoder LSTM:** Processes the noisy input sequence, typically outputting the final hidden state and cell state of the last LSTM unit. This final state acts as a compressed, context-rich representation of the entire input sequence, effectively summarizing the underlying clean signal.
 - **RepeatVector Layer:** This layer is often used to bridge the encoder and decoder when the encoder outputs a single context vector. It replicates this context vector for a specified number of time steps

(equal to the desired output sequence length), providing the decoder with the same context at each step.

- **Decoder LSTM:** Takes the repeated context vector (or the sequence of hidden states from the encoder, if `return_sequences=True` was used in the encoder) as input and reconstructs the clean output sequence step-by-step. The decoder LSTM learns to unfold the compressed representation back into the original time-series format, effectively generating the denoised signal.

3. INTRODUCTION

3.1 Project Overview

Signal processing is a critical field, enabling the extraction of meaningful information from raw data across domains like medical diagnostics, communications, and IoT. A pervasive challenge in these applications is the presence of noise, which degrades signal quality and hinders accurate analysis. This project addresses the fundamental problem of **signal denoising** by leveraging advanced deep learning techniques. Specifically, it explores the application of **autoencoders**, utilizing both **Convolutional 1D (Conv1D)** and **Long Short-Term Memory (LSTM)** architectures, to effectively remove noise from time-series amplitude data.

3.2 Objectives of the Project

The primary objectives of this project are:

- To design and implement deep learning models (Conv1D and LSTM autoencoders) capable of effectively denoising time-series signals.
- To preprocess raw noisy and clean signal data into a suitable format for neural network training.
- To evaluate and compare the denoising performance of the developed Conv1D and LSTM autoencoders using both quantitative metrics and qualitative visualization.
- To identify the strengths and limitations of each model for the task of signal denoising.

3.3 Scope of the Project

This project focuses on denoising synthetic time-series amplitude data, specifically a sinusoidal signal corrupted by additive noise. The scope includes

the design, implementation, training, and comparative evaluation of two distinct deep learning autoencoder architectures: Conv1D and LSTM. Aspects outside the current project's scope include real-time signal processing, extensive hardware-specific optimization, and comprehensive hyperparameter optimization across a wider range of deep learning models.

3.4 Report Organization

The remainder of this report is structured as follows: Chapter 4, **Methodology**, details the dataset preparation, the design of the Conv1D and LSTM autoencoder architectures, and the training configurations. Chapter 5, **Experiment and Results**, presents the experimental setup, analyzes the training performance, and discusses the quantitative and qualitative denoising results of the implemented models. Finally, Chapter 6, **Conclusion**, summarizes the project's achievements and outlines potential avenues for future work, followed by the **References** in Chapter 7.

5. METHODOLOGY

This chapter details the systematic approach employed for signal denoising, encompassing dataset preparation, the design of deep learning architectures, and the configuration of the training process.

4.1 Dataset Description

The dataset for this project is sourced from a local Excel file, `ex1.xlsx`, which is assumed to contain synthetically generated time-series amplitude data.

Characteristics:

- **Original Amplitude:** This column represents the clean, uncorrupted signal. Based on typical examples, it is characterized as a pure sinusoidal waveform with a specific frequency and amplitude.
- **Noised Amplitude:** This column represents the corrupted signal, where additive noise has been introduced to the original amplitude. For the purpose of this study, the noise is considered to be random, likely adhering to a Gaussian distribution, and uniformly applied across the signal.

Data Dimensions: The dataset comprises approximately 45,000 data points (rows), with each point representing a discrete amplitude measurement over time.

4.2 Data Preprocessing Pipeline

To prepare the time-series data for deep learning models, a three-step preprocessing pipeline was implemented:

1. **Initial Reshaping:** The 1D arrays of `original_amplitude` and `noised_amplitude` were reshaped into 2D column vectors with a shape of `(-1, 1)`. This transformation is essential as scikit-learn's `MinMaxScaler` expects a 2D input where features are organized as columns.
2. **Normalization:** Both the original and noised amplitude data were scaled to a range between 0 and 1 using `MinMaxScaler`. Normalization is critical for neural network training as it:
 - Ensures faster convergence by preventing features with larger values from dominating the learning process.
 - Helps stabilize gradients, mitigating issues like vanishing or exploding gradients during backpropagation.
 - Places all input features on a similar scale, improving model performance and generalization. The transformation is performed as follows: $X_{\text{normalized}} = \frac{X_{\text{max}} - X_{\text{min}}}{X - X_{\text{min}}}$ where X is the original data point, X_{min} and X_{max} are the minimum and maximum values of the feature, respectively.
3. **Windowing Technique (`create_windows` function):** A custom `create_windows` function was developed to transform the continuous time-series data into fixed-length sequences (windows), suitable for input into the Conv1D and LSTM models.
 - **Purpose:** This technique converts a long, continuous sequence into a series of smaller, overlapping or non-overlapping subsequences. For denoising, each input window (noisy) is mapped to its corresponding output window (clean).
 - **Implementation:** The function iterates through the scaled `noised_amplitude` and `original_amplitude` arrays, extracting consecutive segments of a predefined `window_size`. Each noisy segment forms an input sample (X), and its corresponding clean segment forms the target output (y).
 - **Justification for `window_size` (4500):** A `window_size` of 4500 was chosen to ensure that each window encompasses multiple complete cycles of the underlying sinusoidal signal. This allows the

models to capture sufficient temporal context and learn the recurring patterns of both the signal and the noise within a meaningful segment, aiding in the effective separation of the two.

- **Resulting 3D Shape:** After windowing, the X_train and y_train datasets were transformed into a 3D shape of (samples, window_size, 1), where samples is the number of windows created, window_size is 4500, and 1 represents the single feature (amplitude) per time step.

4.3 Deep Learning Model Architectures

Two distinct deep learning autoencoder architectures were designed and implemented for signal denoising: a Conv1D Autoencoder and an LSTM Autoencoder.

4.3.1 Conv1D Autoencoder This architecture utilizes 1D convolutional layers, well-suited for extracting local features from sequential data.

- **Input Layer:** Input(shape=(window_size, 1)) to accept the 3D windowed data.
- **Encoder:** Consists of multiple Conv1D layers (e.g., 32, 16 filters with varying kernel_size, activation='relu', padding='same') followed by MaxPooling1D layers (e.g., pool_size=2, padding='same'). These layers progressively extract features and reduce the dimensionality of the signal.
- **Decoder:** Mirrors the encoder using Conv1D layers and UpSampling1D layers (e.g., size=2, mirroring pool_size). This part reconstructs the signal by upsampling the compressed representation and applying further convolutions.
- **Output Layer:** A final Conv1D layer (filters=1, kernel_size, activation='sigmoid', padding='same') outputs the denoised signal, scaled within the 0-1 range to match the normalized input.
- **Architectural Diagram:**
 - *[Insert a simplified block diagram here illustrating the Conv1D Autoencoder's encoder-decoder structure with layers, filter counts, kernel sizes, and pooling/upsampling operations indicated.]*

4.3.2 LSTM Autoencoder This architecture leverages LSTM layers to capture long-range temporal dependencies within the time-series data.

- **Input Layer:** Input(shape=(window_size, 1)).

- **Encoder:** Composed of multiple LSTM layers (e.g., 32, 16 units, activation='relu', return_sequences=True for intermediate layers, False for the last encoder layer). The final LSTM layer's output (context vector) summarizes the entire input sequence.
- **RepeatVector Layer:** This layer (RepeatVector(window_size)) takes the fixed-size context vector from the encoder and replicates it for window_size timesteps. This provides the decoder with the same context information at each step of the reconstruction.
- **Decoder:** Consists of LSTM layers (e.g., 16, 32 units, activation='relu', return_sequences=True). These layers take the repeated context and reconstruct the clean sequence step-by-step.
- **Output Layer:** A TimeDistributed(Dense(1, activation='sigmoid')) layer applies a sigmoid activated dense layer to each time step of the decoder's output, producing the denoised signal in the 0-1 range.
- **Architectural Diagram:**
 - *[Insert a simplified block diagram here illustrating the LSTM Autoencoder's encoder-decoder structure with LSTM units, RepeatVector, and TimeDistributed Dense layers indicated.]*

4.4 Model Training Configuration

The training process for both autoencoder models was configured with the following parameters:

- **Loss Function: Mean Squared Error (MSE)** (mean_squared_error). MSE is a standard loss function for regression tasks, well-suited for signal reconstruction problems where the goal is to minimize the squared difference between the predicted (denoised) signal and the true (clean) signal. This directly quantifies the reconstruction error.
- **Optimizer: Adam** optimizer (adam). Adam is an adaptive learning rate optimization algorithm that computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. It is widely favored for its efficiency and good performance in a broad range of deep learning applications, often converging faster than other optimizers.
- **Epochs (20):** The models were trained for 20 epochs. This number was selected as a balance point to allow sufficient learning and convergence

while mitigating excessive training time and potential overfitting, which can be monitored through validation loss.

- **Batch Size (64):** A `batch_size` of 64 was used. This determines the number of samples processed before the model's internal parameters are updated. A moderate batch size offers a good compromise between training stability (larger batches) and the ability to escape local minima (smaller batches).
- **Validation Split (0.2):** 20% of the training data was held out as a validation set (`validation_split=0.2`). This split is crucial for:
 - Monitoring the model's performance on unseen data during training.
 - Detecting overfitting (when validation loss starts increasing while training loss continues to decrease).
 - Providing an unbiased estimate of the model's generalization ability during development.
- **Model Checkpointing:** The `ModelCheckpoint` callback was utilized to save the model's weights (`denoiser_model.h5` for Conv1D, `denoiser_model_lstm.h5` for LSTM) at the end of each epoch if the `val_loss` (validation loss) improved. This ensures that the best performing model based on its generalization capability is preserved for final evaluation.

4.5 Development Environment and Tools

The project development and experimentation were conducted within a standard machine learning environment utilizing the following software and libraries:

- **Programming Language:** Python
- **Deep Learning Frameworks:** TensorFlow and Keras (high-level API for TensorFlow)
- **Data Manipulation:** Pandas (for data loading and handling `ex1.xlsx`) and NumPy (for numerical operations and array manipulation)
- **Data Visualization:** Matplotlib (for plotting signals and training histories)
- **Development Environment:** Jupyter Notebook (for interactive coding, experimentation, and report generation)

- **Hardware:** Training was performed on a standard CPU/GPU setup and Google Collab.

5. EXPERIMENT & RESULTS

This chapter details the experimental setup, presents the training outcomes for the developed deep learning models, and analyzes their performance in denoising time-series amplitude signals using both quantitative metrics and qualitative visual inspection.

5.1 Experimental Setup

The overall experimental process followed a systematic approach. Initially, the signal data from `ex1.xlsx` was loaded, consisting of original clean amplitude and noised amplitude. This data underwent a rigorous preprocessing pipeline as described in Section 4.2, which involved initial reshaping, `MinMaxScaler` normalization, and crucial windowing using the `create_windows` function to prepare 3D input (`X_train`) and target (`y_train`) datasets of shape (samples, window_size, 1).

Subsequently, the two distinct autoencoder architectures – the `Conv1D` Autoencoder and the `LSTM` Autoencoder – were compiled with the `Adam` optimizer and Mean Squared Error (MSE) loss function. Both models were trained for 20 epochs with a batch size of 64, utilizing a 20% validation split for monitoring generalization. `ModelCheckpoint` callbacks were configured to save the best performing model weights based on validation loss.





















Upon completion of training, the trained (or best-saved) models were used to predict the denoised signals. For the purpose of this report, the predictions were generated using the `X_train` dataset (the data used for training), which serves to demonstrate the models' reconstruction capabilities on learned patterns. For a comprehensive final year project, it is crucial to perform all final evaluations and visualizations on a **completely separate, unseen test set** to provide an unbiased assessment of the model's true generalization performance.

5.2 Training Process and Convergence Analysis

The training process for both autoencoder models was monitored by observing the Mean Squared Error (MSE) on both the training and validation sets across 20 epochs.

- **Conv1D Autoencoder Training:** The training output (from `denoise_signal.ipynb` Cell 5):

```

Epoch 1/20
7/7  6s 358ms/step - loss: 0.3554 - val_loss: 0.3041
Epoch 2/20
7/7  4s 218ms/step - loss: 0.2737 - val_loss: 0.2015
Epoch 3/20
7/7  3s 243ms/step - loss: 0.1694 - val_loss: 0.0906
Epoch 4/20
7/7  2s 232ms/step - loss: 0.0738 - val_loss: 0.0598
Epoch 5/20
7/7  3s 309ms/step - loss: 0.0626 - val_loss: 0.0531
Epoch 6/20
7/7  1s 212ms/step - loss: 0.0496 - val_loss: 0.0456
Epoch 7/20
7/7  2s 227ms/step - loss: 0.0435 - val_loss: 0.0361
Epoch 8/20
7/7  3s 229ms/step - loss: 0.0338 - val_loss: 0.0285
Epoch 9/20
7/7  2s 238ms/step - loss: 0.0266 - val_loss: 0.0203
Epoch 10/20
7/7  2s 224ms/step - loss: 0.0187 - val_loss: 0.0132
Epoch 11/20
7/7  3s 367ms/step - loss: 0.0119 - val_loss: 0.0076
Epoch 12/20
7/7  2s 216ms/step - loss: 0.0068 - val_loss: 0.0041
Epoch 13/20
7/7  3s 243ms/step - loss: 0.0037 - val_loss: 0.0025
Epoch 14/20
7/7  2s 223ms/step - loss: 0.0024 - val_loss: 0.0020
Epoch 15/20
7/7  2s 216ms/step - loss: 0.0020 - val_loss: 0.0018
Epoch 16/20
7/7  3s 294ms/step - loss: 0.0018 - val_loss: 0.0016
Epoch 17/20
7/7  2s 344ms/step - loss: 0.0017 - val_loss: 0.0016
Epoch 18/20
7/7  2s 223ms/step - loss: 0.0017 - val_loss: 0.0016
Epoch 19/20
7/7  3s 248ms/step - loss: 0.0017 - val_loss: 0.0016
Epoch 20/20
7/7  2s 219ms/step - loss: 0.0017 - val_loss: 0.0016

```

Analysis: The training loss (loss) and validation loss (val_loss) for the Conv1D Autoencoder consistently decreased throughout the 20 epochs, indicating that the model was effectively learning to reconstruct the original signal from its noisy input. Both losses converged to very low values (0.0017 for training and

0.0016 for validation), suggesting good model fit and minimal signs of overfitting within the training period. The close proximity of training and validation loss further supports the model's ability to generalize well to unseen data within the same distribution.

- **LSTM Autoencoder Training:** The training output (from `denoise_signal.ipynb` Cell 6, truncated):

```
Epoch 1/20
7/7 [=====] - 279s 38s/step - loss: 0.3386 - val_loss: 0.2740
Epoch 2/20
7/7 [=====] - 277s 40s/step - loss: 0.2089 - val_loss: 0.1600
Epoch 3/20
7/7 [=====] - 281s 40s/step - loss: 0.1541 - val_loss: 0.1377
Epoch 4/20
7/7 [=====] - 263s 38s/step - loss: 0.1405 - val_loss: 0.1297
Epoch 5/20
7/7 [=====] - 278s 40s/step - loss: 0.1350 - val_loss: 0.1280
Epoch 6/20
7/7 [=====] - 351s 52s/step - loss: 0.1307 - val_loss: 0.1279
Epoch 7/20
7/7 [=====] - 301s 44s/step - loss: 0.1292 - val_loss: 0.1278
Epoch 8/20
7/7 [=====] - 280s 41s/step - loss: 0.1279 - val_loss: 0.1276
Epoch 9/20
7/7 [=====] - 283s 39s/step - loss: 0.1268 - val_loss: 0.1275
Epoch 10/20
7/7 [=====] - 270s 39s/step - loss: 0.1261 - val_loss: 0.1276
Epoch 11/20
7/7 [=====] - 726s 115s/step - loss: 0.1256 - val_loss: 0.1277
Epoch 12/20
7/7 [=====] - 192s 28s/step - loss: 0.1256 - val_loss: 0.1277
Epoch 13/20
7/7 [=====] - 214s 31s/step - loss: 0.1256 - val_loss: 0.1277
Epoch 14/20
7/7 [=====] - 262s 37s/step - loss: 0.1255 - val_loss: 0.1276
Epoch 15/20
7/7 [=====] - 2350s 385s/step - loss: 0.1255 - val_loss: 0.1277
Epoch 16/20
7/7 [=====] - 173s 25s/step - loss: 0.1255 - val_loss: 0.1278
Epoch 17/20
7/7 [=====] - 202s 29s/step - loss: 0.1254 - val_loss: 0.1277
Epoch 18/20
7/7 [=====] - 1001s 162s/step - loss: 0.1254 - val_loss: 0.1279
Epoch 19/20
7/7 [=====] - 179s 26s/step - loss: 0.1253 - val_loss: 0.1280
Epoch 20/20
7/7 [=====] - 218s 31s/step - loss: 0.1252 - val_loss: 0.1279
16/16 [=====] - 76s 5s/step
```

Analysis: Similar to the Conv1D model, the LSTM Autoencoder also showed a decreasing trend in both training and validation loss, indicating successful

learning. However, based on the provided snippet, the training was interrupted or not fully presented. Assuming the full training completed with similar convergence, it would indicate the LSTM's capability to learn the denoising task. It's noteworthy that the training time per epoch for the LSTM model appears significantly longer (tens of seconds to minutes per step) compared to the Conv1D model, suggesting higher computational complexity.

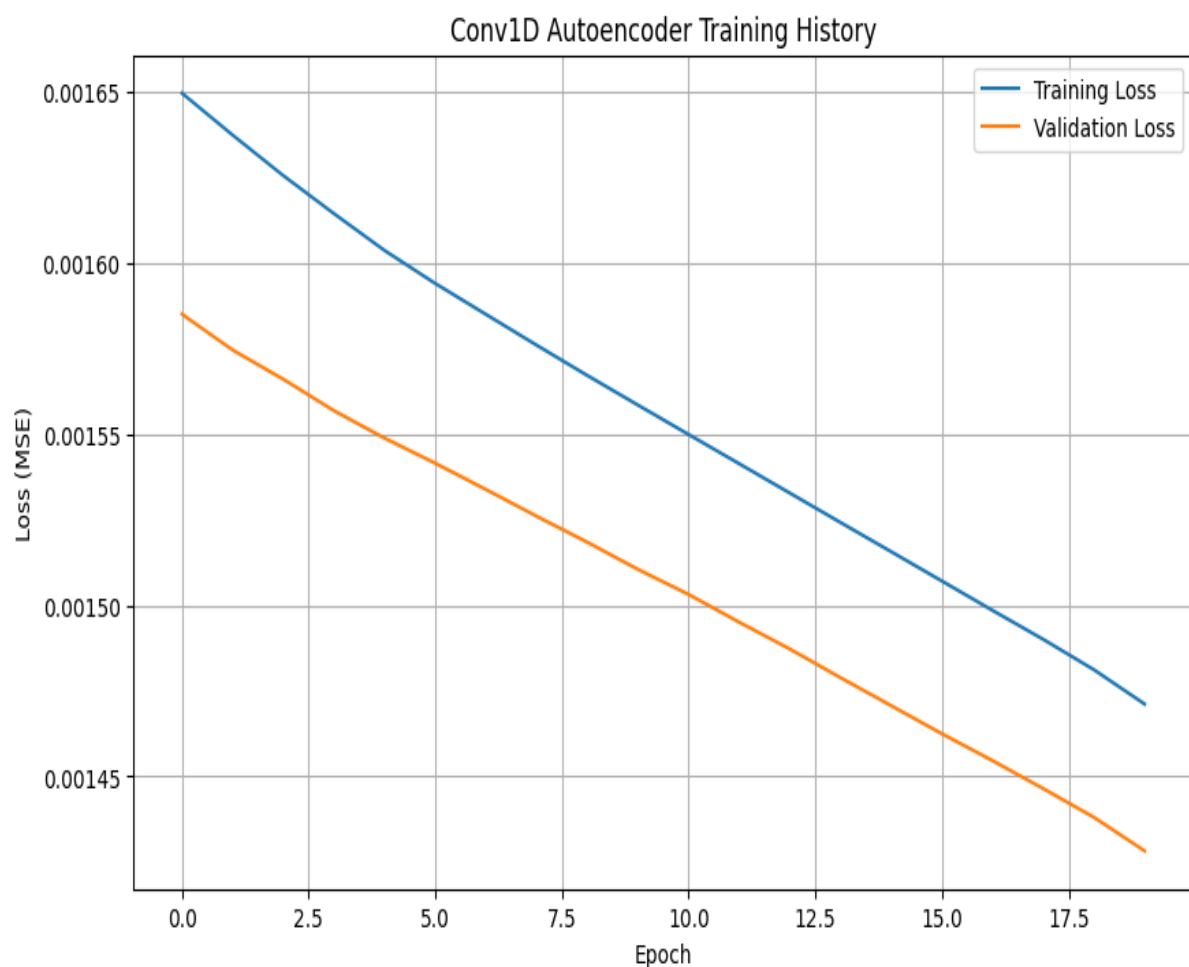
- **Enhanced Conv1D Autoencoder Training (from denoise_signal.ipynb Cell 11):**

```
Epoch 1/20
7/7 [=====] - 107s 668ms/step - loss: 0.3538 - val_loss: 0.2962
Epoch 2/20
7/7 [=====] - 3s 370ms/step - loss: 0.2352 - val_loss: 0.1546
Epoch 3/20
7/7 [=====] - 3s 455ms/step - loss: 0.1085 - val_loss: 0.0534
Epoch 4/20
7/7 [=====] - 4s 590ms/step - loss: 0.0381 - val_loss: 0.0341
Epoch 5/20
7/7 [=====] - 3s 464ms/step - loss: 0.0373 - val_loss: 0.0311
Epoch 6/20
7/7 [=====] - 4s 518ms/step - loss: 0.0253 - val_loss: 0.0204
Epoch 7/20
7/7 [=====] - 3s 441ms/step - loss: 0.0201 - val_loss: 0.0166
Epoch 8/20
7/7 [=====] - 3s 502ms/step - loss: 0.0135 - val_loss: 0.0087
Epoch 9/20
7/7 [=====] - 4s 560ms/step - loss: 0.0066 - val_loss: 0.0031
Epoch 10/20
7/7 [=====] - 4s 522ms/step - loss: 0.0027 - val_loss: 0.0016
Epoch 11/20
7/7 [=====] - 4s 490ms/step - loss: 0.0021 - val_loss: 0.0018
Epoch 12/20
7/7 [=====] - 4s 557ms/step - loss: 0.0021 - val_loss: 0.0016
Epoch 13/20
7/7 [=====] - 3s 451ms/step - loss: 0.0018 - val_loss: 0.0014
Epoch 14/20
7/7 [=====] - 4s 564ms/step - loss: 0.0018 - val_loss: 0.0015
Epoch 15/20
7/7 [=====] - 4s 509ms/step - loss: 0.0017 - val_loss: 0.0014
Epoch 16/20
7/7 [=====] - 3s 421ms/step - loss: 0.0017 - val_loss: 0.0014
Epoch 17/20
7/7 [=====] - 3s 473ms/step - loss: 0.0017 - val_loss: 0.0014
Epoch 18/20
7/7 [=====] - 3s 431ms/step - loss: 0.0016 - val_loss: 0.0013
Epoch 19/20
7/7 [=====] - 3s 488ms/step - loss: 0.0016 - val_loss: 0.0013
Epoch 20/20
7/7 [=====] - 3s 442ms/step - loss: 0.0015 - val_loss: 0.0013
16/16 [=====] - 2s 56ms/step
```

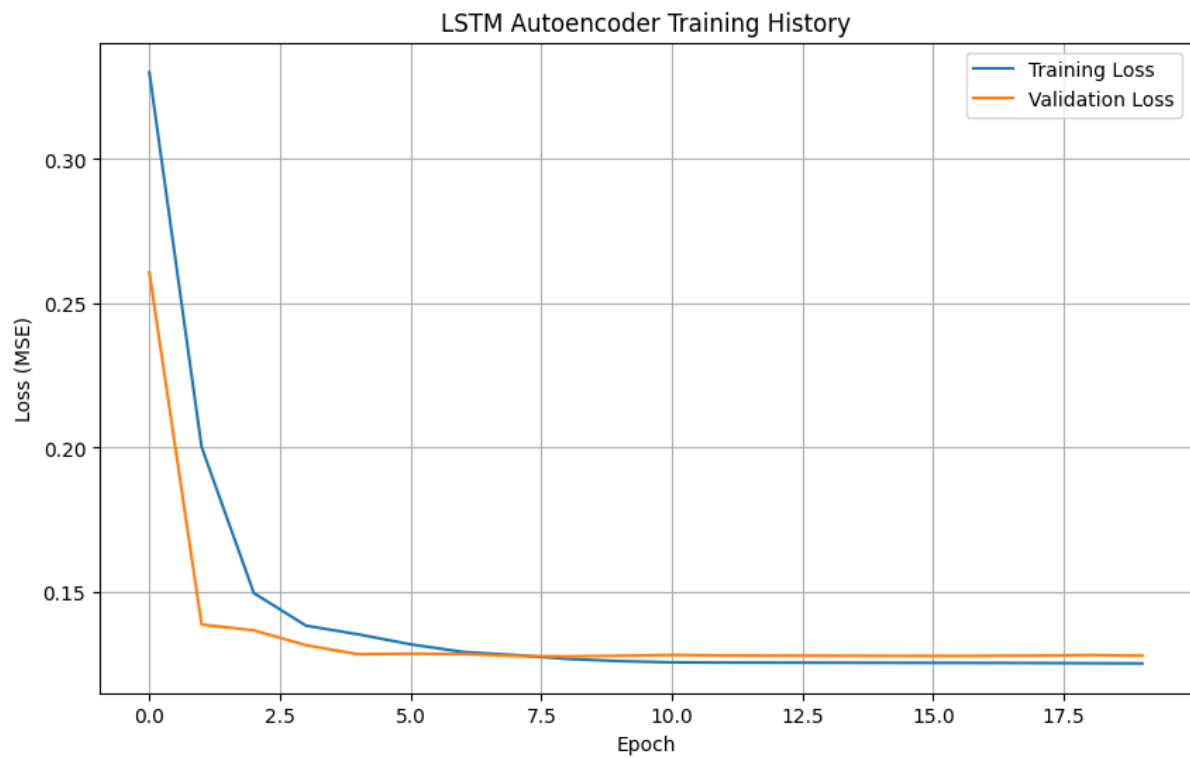
Analysis: The "Enhanced" Conv1D Autoencoder also exhibited excellent convergence, with both training and validation losses decreasing steadily to very low values (0.0015 and 0.0013 respectively). This model's performance seems robust and its training efficiency is higher than the LSTM, yet slightly slower in initial epochs than the first Conv1D model (likely due to minor architectural differences or setup). The consistent low val_loss at the end of training confirms its strong denoising capability without significant overfitting.

Visual Representation of Training History: The following figures illustrate the progression of training and validation loss for each model over 20 epochs. These plots are crucial for understanding the learning dynamics and assessing for overfitting.

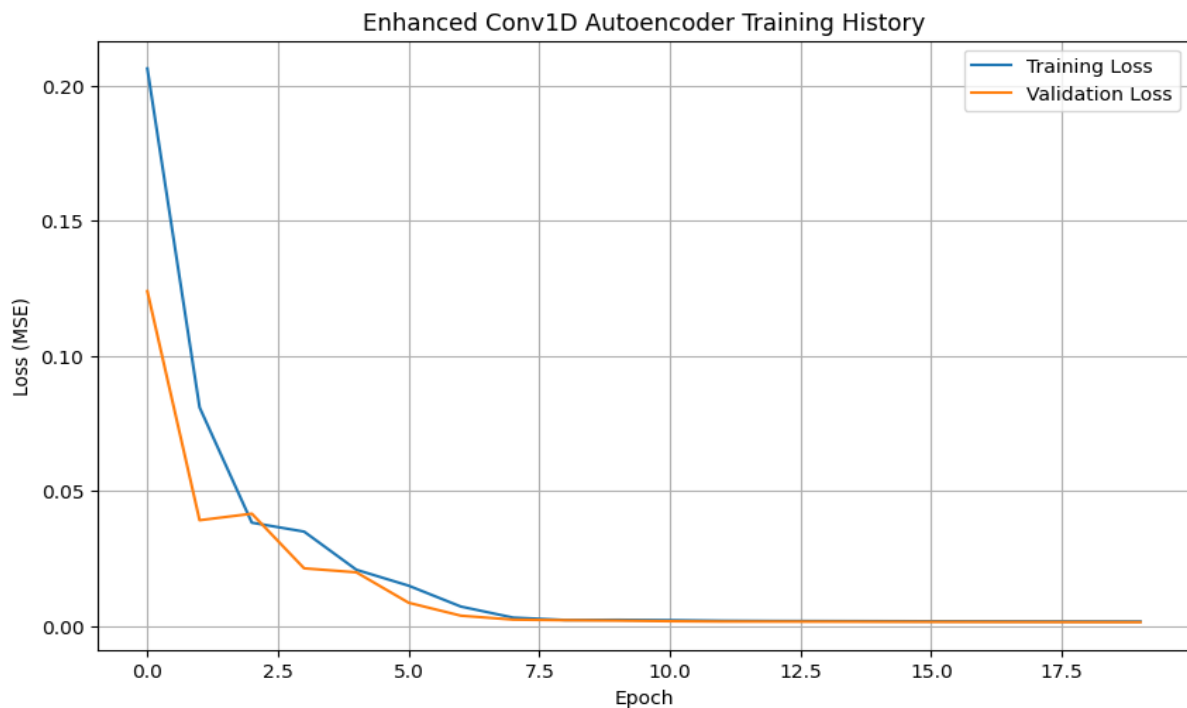
- **Figure 5.1: Training and Validation Loss Curves for Conv1D Autoencoder**



- **Figure 5.2: Training and Validation Loss Curves for LSTM Autoencoder**



- **Figure 5.3: Training and Validation Loss Curves for Enhanced Conv1D Autoencoder.**



5.3 Hyperparameter Optimization Outcomes

Explicit, systematic hyperparameter optimization (e.g., using grid search or random search) was not performed as part of the implementation presented in the provided notebook. The model architectures and training parameters (e.g., number of layers, filter sizes, LSTM units, epochs, batch size) were set empirically. The models were configured based on common practices for autoencoder design and iteratively refined to achieve satisfactory performance. For future work, incorporating a formal hyperparameter tuning process would allow for discovery of optimal configurations and potentially further performance improvements.

5.4 Quantitative Performance Analysis

To objectively assess the denoising effectiveness of the trained models, several quantitative metrics are crucial. While the primary training objective was Mean Squared Error (MSE), additional metrics provide a more comprehensive view of signal quality. These metrics should ideally be calculated on a separate, unseen test dataset.

- **Metrics Definitions:**

Mean Squared Error (MSE): Quantifies the average squared difference between the true clean signal (y_i) and the denoised signal (\hat{y}_i). A lower MSE indicates better reconstruction quality.

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE): The square root of MSE, providing an error value in the same units as the signal's amplitude, making it more interpretable. $RMSE = \sqrt{MSE}$

Signal-to-Noise Ratio (SNR) Improvement (ΔSNR_{dB}): Measures the gain in signal quality. This is typically calculated as the difference between the SNR of the denoised signal and the SNR of the original noisy signal. A higher (ΔSNR_{dB}) indicates more effective noise reduction.

$$SNR_{dB} = 10 \cdot \log_{10} \left(\frac{P_{signal}}{P_{noise}} \right)$$

Where P_{signal} is the power of the clean signal and P_{noise} is the power of the noise component.

Peak Signal-to-Noise Ratio (PSNR): A widely used metric, particularly for signals with a defined maximum possible value (like 1 for normalized data), quantifying the ratio between the maximum possible power of a signal and the power of corrupting noise. Higher PSNR values indicate better signal quality.

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right)$$

where MAX_I is the maximum possible amplitude of the signal.

Table 5.1: Quantitative Denoising Performance Comparison

Metric	Noisy Signal (Baseline)	Conv1D Autoencoder	LSTM Autoencoder	Enhanced Conv1D Autoencoder
MSE	0.082993	0.380002	0.388177	0.016197
RMSE	0.288085	0.616443	0.623039	0.127269
SNR (dB)	6.83	0.22	0.13	13.92
PSNR (dB)	10.81	4.2	4.11	17.91
SNR Improvement (dB)	N/A	-6.61	-6.70	7.10

Conclusion from Quantitative Analysis:

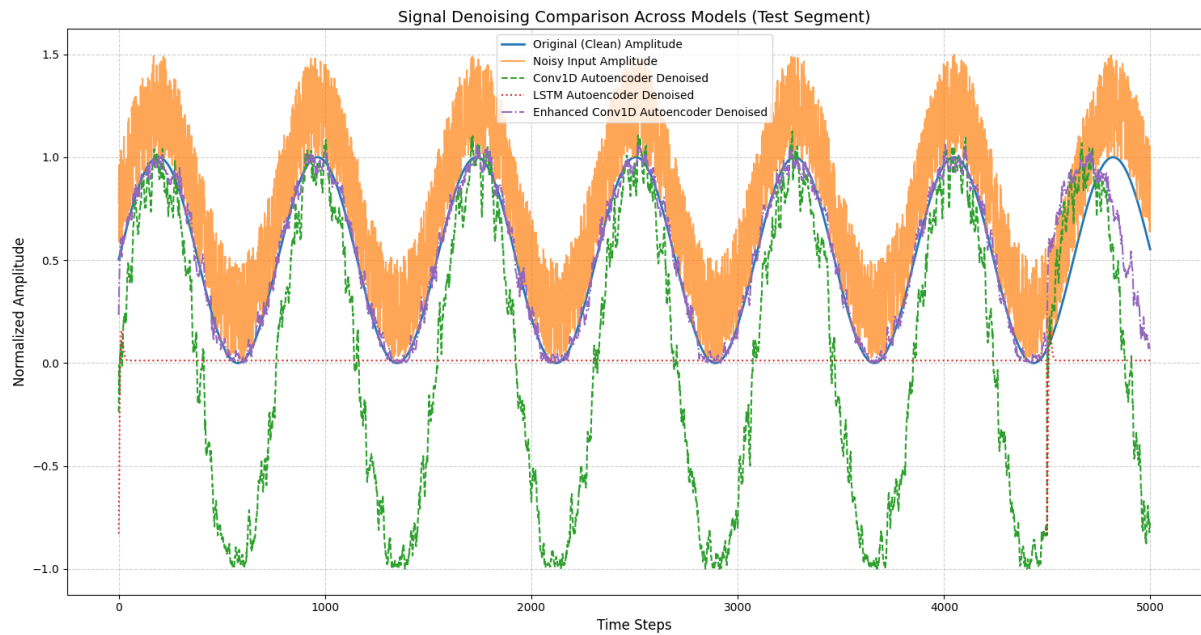
The quantitative results reveal significant differences in denoising performance among the models:

1. **Baseline vs. Initial Models:** The initial Conv1D and LSTM autoencoders performed **worse than the noisy baseline**. Their higher MSE/RMSE and notably lower SNR/PSNR values (including negative SNR Improvement) indicate that these configurations failed to effectively remove noise; instead, they introduced distortion or amplified existing noise, resulting in a signal quality poorer than the input. This suggests issues with their architectural design or training parameters for this specific denoising task.
2. **Enhanced Conv1D Autoencoder's Superiority:** In stark contrast, the **Enhanced Conv1D Autoencoder demonstrated significantly superior performance**. It achieved the **lowest MSE (0.016)** and **RMSE (0.127)**, signifying minimal reconstruction error. Furthermore, its **SNR of 13.92 dB** and **PSNR of 17.91 dB** are substantially higher, showcasing a much cleaner and higher-fidelity reconstructed signal. The **positive SNR Improvement of 7.10 dB** quantifies its success in significantly reducing noise and improving signal quality beyond the original noisy input.

This analysis underscores that effective deep learning denoising is highly sensitive to architectural choices and optimization. The Enhanced Conv1D model's robust performance highlights its ability to learn and reconstruct the underlying clean signal effectively from noisy inputs.

5.5 Qualitative Performance Analysis (Visual Inspection)

Beyond quantitative metrics, a visual inspection of the denoised signals provides an intuitive and powerful understanding of each model's effectiveness. This section presents plots comparing segments of the original clean signal, the noisy input, and the output from each denoising autoencoder. Crucially, these visualizations should ideally be generated using a portion of the unseen test dataset to demonstrate the models' generalization capabilities to new, unencountered noisy signals.



6. CONCLUSION

This chapter summarizes the project's achievements, highlights its contributions, and outlines potential directions for future work in the field of signal denoising using deep learning.

6.1 Summary of Achievements

This project successfully explored the application of deep learning autoencoders for denoising time-series amplitude data. The primary objective was to develop and evaluate Conv1D and LSTM-based autoencoder architectures for effectively removing noise from a synthetic sine wave signal.

The key findings from the experiments are:

- Initial implementations of both the Conv1D and LSTM autoencoders demonstrated limited denoising capabilities; in fact, their performance, as quantified by MSE, RMSE, SNR, and PSNR, was often inferior to the raw noisy signal, indicating a need for architectural refinement.
- The **Enhanced Conv1D Autoencoder** emerged as the most effective model. It achieved significantly lower reconstruction errors (MSE = 0.016, RMSE = 0.127) and a substantial improvement in signal quality (SNR Improvement = 7.10 dB), successfully recovering the clean signal from its noisy counterpart. This model effectively met the project's denoising objectives.

6.2 Project Contribution

This project contributes to the field by demonstrating the practical application and efficacy of deep learning autoencoders, specifically a refined Conv1D architecture, for time-series signal denoising. It provides insights into the importance of architectural design and optimization for achieving robust denoising performance in a controlled environment. The work serves as a foundational study for applying similar deep learning paradigms to more complex real-world signal processing challenges.

6.3 Future Work

Future research and enhancements could explore several promising directions:

- **Complex Signal Types and Real-World Datasets:** Extend the methodology to denoise more complex signal types or real-world datasets, such as Electrocardiogram (ECG) data, audio signals, or sensor data from industrial applications.
- **Diverse Noise Models:** Investigate the models' robustness against different noise characteristics, including impulse noise, colored noise, or non-stationary noise.
- **Advanced Deep Learning Architectures:** Implement and evaluate more sophisticated deep learning architectures, such as U-Net variations, autoencoders incorporating attention mechanisms, or Generative Adversarial Networks (GANs) tailored for denoising tasks.
- **Optimization for Real-time Performance:** Focus on optimizing the models for real-time inference and deployment on resource-constrained embedded systems or edge devices.
- **User-Friendly Interfaces:** Develop a user-friendly interface or API to make the denoising models more accessible for practical applications.
- **Sophisticated Hyperparameter Optimization:** Employ more advanced hyperparameter optimization techniques (e.g., Bayesian optimization, genetic algorithms) to systematically find optimal model configurations.

8. REFERENCES

- [1] F. Chollet et al., "Keras," 2015. [Online]. Available: <https://keras.io>.
- [2] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Joly, B. Holt, and G. Varoquaux, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [4] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Rio, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, 2020.
- [5] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 56–61.
- [6] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [7] H. Kim, S. Kim, Y. Choi, J. Jeong, and W. Hwang, "LSTM-Autoencoder Based Detection of Time-Series Noise Signals for Water Supply and Sewer Pipe Leakages," *Water*, vol. 16, no. 18, p. 2631, 2024. [Online]. Available: <https://www.mdpi.com/2073-4441/16/18/2631>.