Discovering anomalies in financial markets using unsupervised learning

Cozmina Scorobete, Carmen Crăciun, Robert Vlășan
Facultatea de Matematică și Informatică
Universitatea de Vest, Timișoara, România

January 14, 2025

Abstract

The rapid growth of the theology made it crucial to have systems that are able to predict anomalies behavior in stock markets. Machine Learning (ML) and Deep Learning (DL) techniques have proven to be effective in anomaly detection. We decided to explore how the autoencoders can be used to detect anomalies in high dimension data-sets like the stock market, by training 2 models one that has synthetic data from Lorenz Attractor and another one that has synthetic data from Van der Pol.

Contents

1 Introduction

In the current time financial markets are essential for the economic impact, due to it is a critical task to identify anomalies(chaotic behavior, unexpected shifts or critical transitions). In order to solve this issue machine learning (ML) became an essential toll for understanding complex dynamics in this field. Anomalies in this field may lead to economic crises, fraud, or market inefficiencies, however early detection could allow to mitigating potential or exploiting emerging opportunities.

To solve this critical problem, models like Long Therm Short Memory(LSTM) based autoencoders excel in detecting anomalies, they capture complex spatial and temporal relationships. However this is not an easy task as a result of the high dimension of the data and the noise in it, it may also be the non-stationary nature of the markets as those are always fluctuating. This fact may also lead to a greater difficulty in distinguishing anomalies from regular fluctuations.

1.1 Research Question

Those mentioned before lead us to our main focus of this research. How can machine learning techniques, particularly unsupervised learning methods such as autoencoders, be applied to detect anomalies in complex dynamical systems like financial markets, while taking into account challenges such as high dimension or noise?

1.2 Introduction to Anomaly Detection in Financial Markets

Economic crises, fraud, or market inefficiencies are some of the problems that may appear due to anomalies, this lead to ML approaches increase in popularity. This is a result of traditional methods' inability to handle the complexity of financial data. In order to make informed decisions and to have market stability this field is supported by deep learning models(DL), such as autoencoders which are excellent at revealing hidden patterns and deviations.

1.3 Traditional Methods of Anomaly Detection

Traditional methods like k-means clustering, moving averages, and Z-scores analysis are commonly used because they are simpler and handle well basic datasets. However, these methods are limited, and can not address the challenges in high-dimensional datasets, nonlinear and time-dependent datasets. In [TBJS20] it is highlighted that such approaches do not perform well on complex and dynamic environments because they rely on fixed criteria making them less capable of capturing the anomalies.

1.4 Unsupervised Learning Techniques

Autoencoders are used to train an autoencoder to learn the underlying structure of the time-series data by compressing it into a low-dimensional latent space. Anomalies, such as chaotic transitions or irregular behavior, are expected to have a high reconstruction error as they deviate from the learned normal patterns. We will also apply clustering to the latent representations learned by the autoencoder or directly to the raw data to group similar behaviors in the system. Transitions between clusters will indicate significant behavioral changes, potentially representing anomalies or chaotic events.

1.5 Deep Learning and Autoencoders

Deep learning has significantly advanced anomaly detection by allowing models to learn complex, nonlinear relationships in high-dimensional datasets. Autoencoders are efficient for anomaly detection as they compress the data into latent representations and reconstruct it, with anomalies being isolated. In [Mic22], it is stated that autoencoders excel in identifying patterns that traditional methods fails to classify. There are some variants like Convolutional Autoencoders (CAEs) and LSTM-based autoencoders that develop this taking into account temporal and spatial features.

2 Methodology

We decided to analyze the difference in performance for a model trained with synthetic data generated with Lorenz Attractor as well as one with Van der Pole the synthetic data will be combine with real data from stock markets. At fer the training of the model and verification of the accuracy we used the data to predict anomalies in stock markets.

2.1 Used Data

In order to have an model accustomed with the anomalies we decided to use 2 types of data. Real data from stock markets as well as synthetic data (Lorenz Attractor or Van der Pol) that we know it has anomalies. We combined the real data with the synthetic data in order to train the model. However, because of the high amount of anomalies in the Lorenz Attractor we decided to test it as well with the Van der Pol to verify if there are changes in the performance.

2.1.1 Collecting Real Data

The primary dataset used in the project combines stock market data collected using the ‘yfinance‘ library. This is a useful tool for gathering historical stock prices, transaction volumes, and other financial information from Yahoo Finance. This dataset includes historical stock prices, transaction volumes, and other relevant financial variables.

```
# Download data for each company
for ticker in ticker_symbols:
    print(f"Fetching data for {ticker}...")
    data = yf.download(ticker, start="2018-01-01", end="2024-01-01")
    data_dict[ticker] = data  # Store the DataFrame in the dictionary

    # Save the data to a CSV file
    output_file = os.path.join(output_dir, f"{ticker}_data.csv")
    data.to_csv(output_file)
    print(f"Data for {ticker} saved to {output_file}.")
```

Listing 1: Get data from yfinance

2.1.2 Generation of the synthetic data

Synthetic Data Generated via Lorenz System: The Lorenz attractor is a very popular system in the field of chaos theory, it shows the the behavior of a dynamical system with three nonlinear differential equations.These equations were originally developed by Edward Lorenz to model atmospheric convection. In [Wil79] is presented the system has a sensitive dependence on initial conditions, causes small changes to lead to vastly different outcomes. This chaotic nature makes the Lorenz attractor a good example for complexity in dynamical systems.

The **lorenz** function defines the 3 equations computing the rate of change of the system's three state variables (x,y,z) based on the parameters (sigma, beta, rho).

The **generate_lorenz_data** function uses a numerical solver to simulate the system over a specified time span and step size, it returns the points corresponding to the states of the attractor.

```
def lorenz(t, state, sigma, beta, rho):
    x, y, z = state
    dx = sigma * (y - x)
    dy = x * (rho - z) - y
    dz = x * y - beta * z
    return [dx, dy, dz]

# Generate synthetic chaotic data
def generate_lorenz_data(sigma=10, beta=8/3, rho=28, t_span=(0, 50), dt=0.01, initial_state=[1.0, 1.0, 1.0]):
    t = np.arange(t_span[0], t_span[1], dt)
    solution = solve_ivp(lorenz, t_span, initial_state, t_eval=t, args=(sigma, beta, rho))
    return t, solution.y.T
```

Listing 2: Generate data with Lorenz

In order to control the system we decided to inject some noise into the data with the following code, it uses both clustered and random anomalies. In order to mimic anomalies the number of those is randomized. We also introduced 3

types of anomalies randomly: spikes(sudden large deviations), offsets(systematic shift), and flat lines (constant zero values).

```
# Step 1: Clustered anomalies
    if clustered:
        cluster_size = max(1, num_anomalies // 10)
        start_indices = np.random.choice(np.arange(num_points -
    cluster_size), size=10, replace=False)
        for start in start_indices:
            for i in range(cluster_size):
                idx = start + i
                states_with_anomalies[idx] += scale * np.random.
    randn(*states[idx].shape) * np.random.uniform(1, 2)
                labels[idx] = 1

    # Step 2: Random anomalies
    remaining_anomalies = num_anomalies - (10 * cluster_size if
    clustered else 0)
    anomaly_indices = np.random.choice(np.arange(num_points), size=
    remaining_anomalies, replace=False)
    for idx in anomaly_indices:
        anomaly_type = np.random.choice(["spike", "offset", "
    flatline"])
        if anomaly_type == "spike":
            states_with_anomalies[idx] += scale * np.random.randn(*
    states[idx].shape)
        elif anomaly_type == "offset":
            states_with_anomalies[idx] += scale * np.random.uniform
    (0.5, 2.0)
        elif anomaly_type == "flatline":
            states_with_anomalies[idx] = np.zeros_like(states[idx])
        labels[idx] = 1
```

Listing 3: Introduce anomalies

Synthetic Data Generated via Van der Pol: Using the Van der Pol approach, synthetic data was created in order to better examine how anomalies behaved in a controlled setting.

The following code sequence generates noisy synthetic data using the Van der Pol oscillator,

```
def van_der_pol(t, z, mu):
    x, y = z
    dxdt = y
    dydt = mu * (1 - x**2) * y - x
    return [dxdt, dydt]

# Parameters
mu = 1.0  # Nonlinearity parameter
time_span = (0, 100)  # Time range for integration
initial_conditions = [2.0, 0.0]  # Initial conditions [x0, y0]
t_eval = np.linspace(time_span[0], time_span[1], 5000)  # Time
    points to evaluate

# Solve the system
solution = solve_ivp(van_der_pol, time_span, initial_conditions,
    t_eval=t_eval, args=(mu,))
```

```
15
16 # Extract the solution
17 x, y = solution.y
18
19 # Add noise to the data
20 noise_level = 0.1  # Adjust noise level
21 x_noisy = x + noise_level * np.random.randn(len(x))
22 y_noisy = y + noise_level * np.random.randn(len(y))
```

Listing 4: Generate Synthetic data with Van der Pol

After the data was collected we used it to create sequenced to "feed" the model. We trained the model on a combination of stock market data and synthetic data, after that we tested the model only on the synthetic data witch is labeled to check the model's accuracy and other metrics.

In order to create the sequenced we used the following code which normalizes the data using the 'MinMacScaler' to scale the values to a range and it divides the data into fixed-length sequences.

```
# Read the CSV file
    data = pd.read_csv(file_path)

    # Check that the dataset has 'x' and 'y' columns
    if not {'x', 'y'}.issubset(data.columns):
        raise ValueError(f"File {file_path} does not have 'x' and '
    y' columns.")

    # Normalize the 'x' and 'y' variables using MinMaxScaler
    scaler = MinMaxScaler()
    data[['x', 'y']] = scaler.fit_transform(data[['x', 'y']])

    # Ensure that we have enough data for the specified sequence
    length
    if len(data) < seq_length:
        raise ValueError(f"Data has fewer than {seq_length} rows,
    cannot create sequences.")

    # Create sequences of length 'seq_length'
    sequences = []
    for i in range(len(data) - seq_length):
        seq = data[['x', 'y']].iloc[i:i + seq_length].values
        sequences.append(seq)
```

Listing 5: Process Data

2.2 Arhitecutre of the Model

We have used an LSTM-base sequence-to-sequence autoencoder which is design for time series data. It has 3 stacked LSTM layers with dimension that is reduced for each layer to ensure that it is capturing temporal relationships.

The decoder is the same type with 3 stacked LSTM layers with dimension that is reduced for each layer. The output layer uses TimeDistributed with a Dense unit that outputs for every time stem in order to reconstruct the input sequence.

We have also create a **applay_smooth** to add more anomalies because the performance of the model before it was very low. The **sooth_predictions** function is used to make the models predictions less noisy.

```
def build_autoencoder(input_dim, latent_dim=32):

    inputs = Input(shape=(input_dim, 1))
    # Encoder with increased layers
    encoded = LSTM(128, activation='relu', return_sequences=True)(
    inputs)
    encoded = LSTM(64, activation='relu', return_sequences=True)(
    encoded)
    encoded = LSTM(latent_dim, activation='relu', return_sequences=
    False)(encoded)

    # Latent Representation
    latent = RepeatVector(input_dim)(encoded)

    # Decoder with increased layers
    decoded = LSTM(latent_dim, activation='relu', return_sequences=
    True)(latent)
    decoded = LSTM(64, activation='relu', return_sequences=True)(
    decoded)
    decoded = LSTM(128, activation='relu', return_sequences=True)(
    decoded)
    outputs = TimeDistributed(Dense(1))(decoded)

    # Autoencoder model
    autoencoder = Model(inputs, outputs)
    autoencoder.compile(optimizer=Adam(learning_rate=0.001), loss='
    mse')
    return autoencoder


def apply_smote(train_data):


    print("Applying SMOTE for oversampling anomalies...")
    labels = np.array([1 if np.random.rand() < 0.2 else 0 for _ in
    range(len(train_data))])
    train_data_flat = train_data.reshape(train_data.shape[0], -1)
    # Flatten for SMOTE
    smote = SMOTE(sampling_strategy=0.3, random_state=42)
    smote_data, _ = smote.fit_resample(train_data_flat, labels)
    return smote_data.reshape(smote_data.shape[0], train_data.shape
    [1], 1)

def smooth_predictions(predictions, window_size=5):

    smoothed = np.convolve(predictions, np.ones(window_size), 'same
    ') / window_size
    return (smoothed > 0.5).astype(int)
```

Listing 6: Model

2.3 Evaluation

To evaluate the performance of the models we used metrics like precision, recall, and F1 scores based on known anomalies in the datasets (for both synthetic and real-world data) as well as confusion matrix and the ROC Curve. We load the sequences and the ground truth labels and the we predict the anomalies.

```
# Load test sequences
    test_sequences = np.loadtxt(test_sequences_file, delimiter=",")
    test_sequences = test_sequences.reshape((test_sequences.shape
    [0], -1, 1))  # Reshape to (samples, timesteps, features)

    # Load ground truth labels
    ground_truth_data = pd.read_csv(ground_truth_file)
    labels = ground_truth_data["anomaly"].values

    # Ensure label alignment
    if len(test_sequences) != len(labels):
        labels = labels[-len(test_sequences):]  # Align labels with
     test sequences if needed

    # Reconstruct the sequences using the autoencoder
    reconstructed_data = autoencoder.predict(test_sequences)

    # Compute reconstruction errors
    errors = np.mean(np.square(test_sequences - reconstructed_data)
    , axis=(1, 2))

    # Determine the threshold dynamically if not provided
    if threshold is None:
        threshold = np.percentile(errors, 90)
        print(f"Using dynamically calculated threshold: {threshold
    }")

    # Predict anomalies based on the threshold
    predicted_anomalies = (errors > threshold).astype(int)
```

Listing 7: Validation of model

2.4 Tools and Frameworks

Python with the frameworks TensorFlow, Keras, Pandas, NumPy, Matplotlib, sklearn.metrics , imblearn.over_sampling. And the yfinance lybrary.

3 Related work

3.1 Anomaly Detection with Robust Deep Autoencoders

In their paper [ZP17], to overcome the typical problem of handling noisy and outlier-infested data, Zhou and Paenroth provide a unique approach for identifying anomalies that makes use of deep learning. In the presence of abnormalities, traditional deep-denoising autoencoders perform much worse. However, they are quite good at learning representations from clean data. The authors suggest

the Robust Deep Autoencoder (RDA) to overcome this restriction, which makes conventional autoencoders more resilient.

The authors further extend their approach by introducing the Group Robust Deep Autoencoder (GRDA), in order to handle more complicated situations in which the data include both random anomalies and organized corruption. The advantage of this approach is that it can distinguish between the types of noise. This would be helpful in real world applications where the noise may not be random.

The performance of the model is proven by extensive experiments on benchmarks datasets. They evaluated it in contrast with other state-of-the-art anomaly detection methods, and the results show that their approach performs better in therms of accuracy an robustness.

3.2 A study of deep convolutional auto-encoders for anomaly detection in videos

Perlin and Lopes [RLL18] address the significant limitations of traditional handcrafted features used in anomaly detection for video surveillance. Methods like social force models, dense trajectories, and dynamic textures require a broad domain-specific knowledge, which is hard to us on different applications. This reliance on a prior knowledge often lead to poor performance. The problem with those traditional approaches is the need of predefined features and assumptions, leading to and approach that is automatic and can generalize features from data. They propose the use of

Convolution al Auto-Encoders(CAEs) in order to identify anomalies in videos. They have used reconstruction error as anomalies, but also tested how more information like motion data added to the frames affected the model's performance. A series of experiments were conducted and the result show that if appearance features are added to the models then the performance increases.

3.3 Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction

In this paper [SY14], appears the idea of detecting anomalies in datasets with nonlinear relationships using autoencoders. They use them to reduce the dimension of the data while still maintain complex relations. They tested the system on synthetic data(Lorenz System) as well as on real world data from spacecraft tenantry data. They have compared the performance of the model with the linear PCA for detecting anomalies and the proposed model outperformed linear PCA. This study also indicates the efficiency of the autoencoder compared to that of the PCA. By analyzing all they demonstrate the efficiency of the model to characterize the anomalies.

3.4 Unsupervised Anomaly Detection in Time Series Using LSTM-Based Autoencoders

This paper [PLV19] investigates unsupervised anomaly detection in time series using Long Short-Term Memory (LSTM)-based autoencoders. The difficulty in this field is challenging due to the unpredictable nature of anomalies and the lack of labeled datasets. The traditional methods (lustering or Gaussian distribution-based techniques) do not seem to categorize all the dependencies.

The authors propose using autoencoders in order to reduce dimension. The study focuses on the advantage of LSTM layers in keeping temporal information. The method was tested on artificial datasets as well as on DCASE dataset and it shown an accuracy of 87%, but only after applying smoothies and response adjustment . The accuracy shows how hard is to optimize a model, the biggest challenge being the selection of hyper-parameters.

4 Experiments and Results

Below are provided the result of the experiment and an overview of the Evaluation method in depth.

4.1 Evaluation and Metric

In order to evaluate the performance of the anomaly detection for both models we incorporated multiple metrics.

- Reconstructive Error: quantifies the difference between the original input and the model's reconstructed output
- Precision: proportion of detected anomalies that are actual anomalies
- Recall: the number of anomalies successfully detected
- F1-Score: combines precision and recall
- ROC-AUC: decides if the model discriminates between normal and anomalous data
- Training and Validation Loss curves: analyzes the stability of the model during training

4.2 Results for Lorenz and Van der Pol Models

4.2.1 Result for Lorenz

The Lorenz model had a precision of 90% for normal data and 88% for anomalies, this indicates that most detected anomalies were correctly identified. On contrast the recall for anomalies was lower at 50% this means that some anomalies

were not detected. The ROC-AUC score was 0.85 this shows strong discrimination between normal and anomalies data.

The reconstruction error plot showed a spike at the beginning of the dataset but this is inconclusive because it could be from a specific anomaly or a data problem.

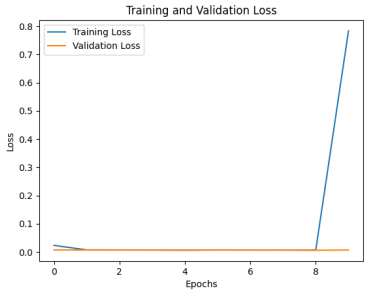
The training loss showed instability, diverging in later eposes, this may suggest over fittings or difficulty in learning certain patterns, but the validation loss remained stable this indicates consistent performance on unseen data.

4.2.2 Result for Van der Pol

The Van der Pol model had a precision of 89% for normal data and 79% for anomalies, the recall for anomalies was lower at 45% this means that some anomalies were not detected. The ROC-AUC score was 0.84 this is close to Lorenz model but shows a minor drop in discrimination capability.

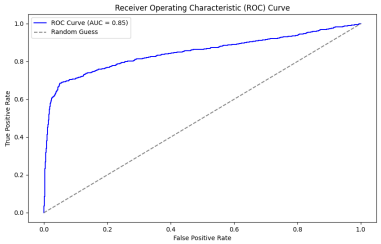
The reconstruction error plot showed no significant spike with a more consistent error distribution, the lack of spikes indicates subtler patterns of anomalies.

The training loss showed stability for both training loss validation loss indicating a stable learning and no over fitting.

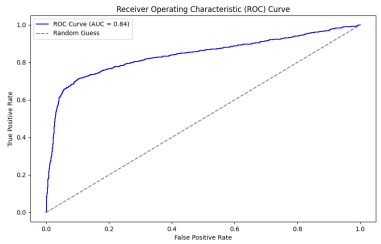


(a) Training and validation loss Lorenz

(b) Training and validation loss Van der Pol



(a) ROC curves Lorenz



(b) ROC curves Van der Pol

4.3 Analysis

- Even tho the Lorenz Model continues to demonstrated to be better at detecting anomalies in chaotic datasets the Van der Pol model has a lower performance doe to the fact that it has added difficulty of detecting anomalies in a periodic nature.
- The Lorenz dataset has a more chaotic nature which lead to distinct anomaly patterns that are easier to identify, the periodic dynamic of Van der Pol system makes anomalies less prominent and harder to identify.

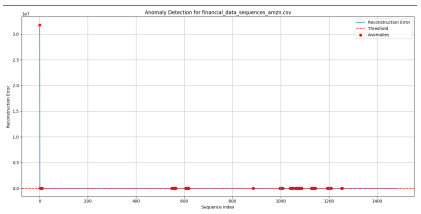
This demonstrated that for the Lorenz Model can represent unpredictable events such as market crashes, rapid price changes or unexpected spikes. However the Van der Pole has it's straights due to the cyclical nature of the model it can represent seasonal variations, trading cycles or gradual shifts in market inefficiencies.

Metric	**Lorenz Model**	**Van der Pol Model**
Precision (Normal)	0.90	0.89
Precision (Anomaly)	0.88	0.79
Recall (Normal)	0.99	0.97
Recall (Anomaly)	0.50	0.45
F1-Score (Normal)	0.94	0.93
F1-Score (Anomaly)	0.64	0.57
Accuracy	0.90	0.88
ROC-AUC	0.85	0.84
Training Loss (Final Epoch)	Diverges	Converges
Validation Loss	Stable	Stable
Reconstruction Error	High spike observed	No high spike observed

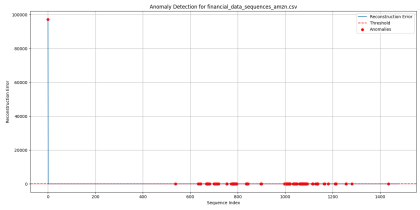
Table 1: Comparison of Lorenz and Van der Pol Models

4.4 Predictions on Actual Data From Stock Market

In the graphs shown below can be seen the actual predictions of each model on the actual stock markets. The Lorenz Model has a significant spike in the beginning the anomalies detected there are more frequent, however Van der Pol model shows a stable reconstruction of errors.

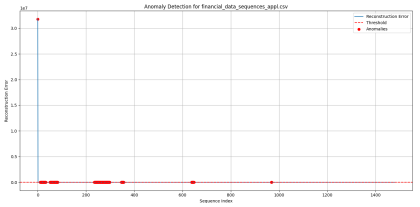


(a) Lorenz Model

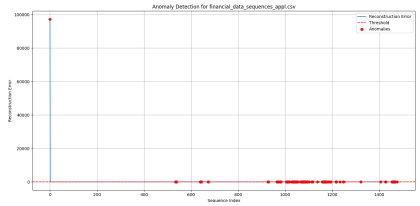


(b) Van der Pol Model

Figure 3: Amazon Data

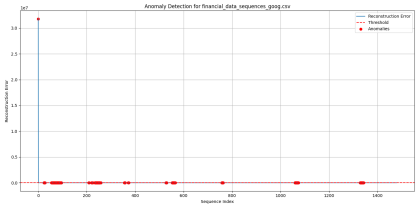


(a) Lorenz Model

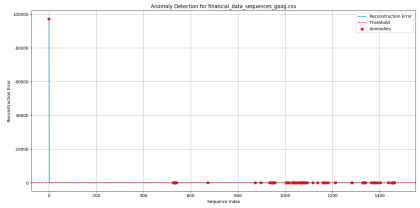


(b) Van der Pol Model

Figure 4: Apple Data

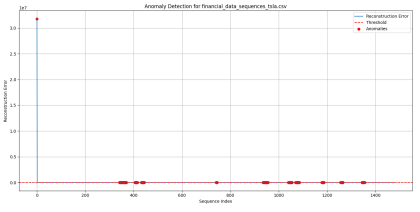


(a) Lorenz Model

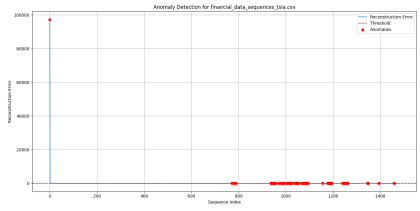


(b) Van der Pol Model

Figure 5: Google Data



(a) Lorenz Model



(b) Van der Pol Model

Figure 6: Tesla Data

5 Conclusion and future directions

5.1 Conclusion

To summarize we have explored anomaly detection in time series systems using unsupervised learning. We have focused on autoencoders and we were able to demonstrate that using the Lorenz Attractor and Van der Pol datasets are 2 competent systems in anomaly detection. The Van der Pol models showed stable learning but reduced performance due to the cyclic patterns, while the Lorenz Model it was highly effective more than the Van der Pole. The result of this research showed how the accuracy of the model is impacted by the data set's properties such as periodic and chaotic.

5.2 Future Work

This research is promising but there are some aspects that need to be taken into account for future work, those being combine models Lorenz and Van der Pol, or implementing a new version of the autoencoder because the one presented in this paper is only a traditional approach.

References

[Mic22] Umberto Michelucci. An introduction to autoencoders. *arXiv preprint arXiv:2201.03898*, 2022.

[PLV19] Oleksandr I Provotar, Yaroslav M Linder, and Maksym M Veres. Unsupervised anomaly detection in time series using lstm-based autoencoders. In *2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, pages 513–517. IEEE, 2019.

[RLL18] Manassés Ribeiro, André Eugênio Lazzaretti, and Heitor Silvério Lopes. A study of deep convolutional auto-encoders for anomaly detection in videos. *Pattern Recognition Letters*, 105:13–22, 2018.

[SY14] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, pages 4–11, 2014.

[TBJS20] Srikanth Thudumu, Philip Branch, Jiong Jin, and Jugdutt Singh. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, 7:1–30, 2020.

[Wil79] Robert F Williams. The structure of lorenz attractors. *Publications Mathematiques de l'IHES*, 50:73–99, 1979.

[ZP17] Chong Zhou and Randy C Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 665–674, 2017.