Unsupervised Anomaly Detection on NASA Bearing Data using Autoencoders

Abstract

This project focuses on the use of unsupervised learning for detecting anomalies in vibration signals from machinery. Using the NASA IMS Bearing Dataset, I have implemented an autoencoder model to learn patterns of normal behavior and detect faults based on reconstruction error. This approach simulates predictive maintenance systems, helping prevent machine failure without requiring labeled data. Various visualization and evaluation techniques were applied to validate the model's performance.

Introduction

In industrial systems, early fault detection is vital to avoid breakdowns and minimize downtime. Traditional methods often rely on labeled datasets, which are difficult to obtain in real-world scenarios. This project proposes an unsupervised anomaly detection system using autoencoders. The system learns to reconstruct normal signals and flags deviations as potential anomalies.

Dataset Overview

We use the NASA IMS Bearing Dataset (Set 1), which includes high-frequency vibration signal recordings. Each file contains 20,480 samples per second from 8 channels (four bearings, each with horizontal and vertical readings). The dataset was preprocessed by normalizing the signals and slicing them into 100-sample windows for model input.

Methodology

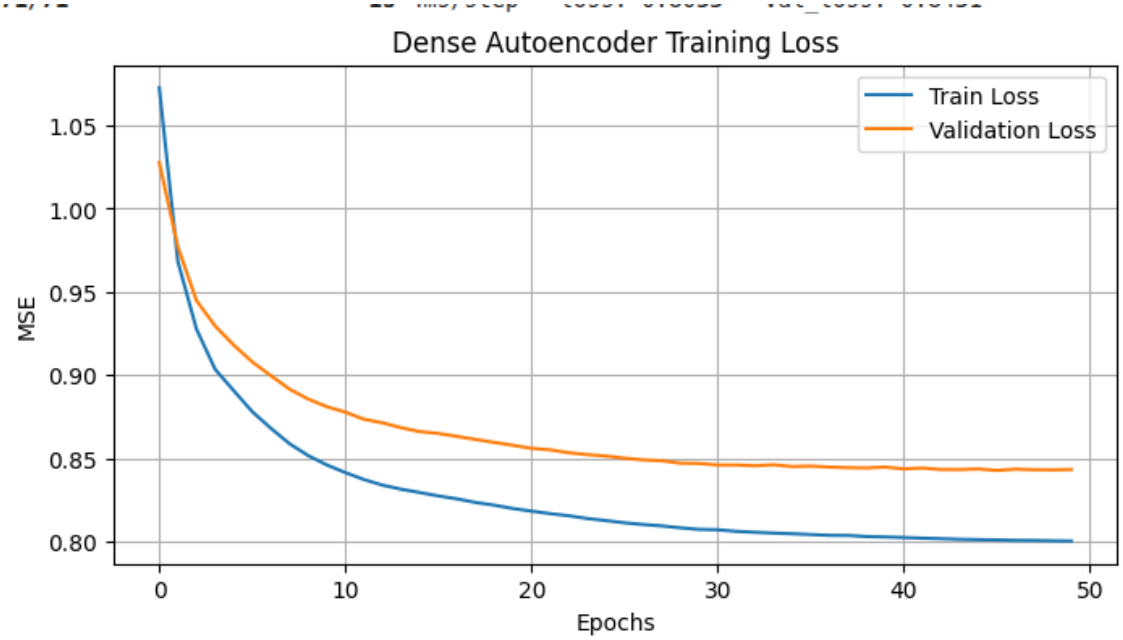
The core methodology involves training a deep autoencoder on normal vibration data. The trained model is then used to reconstruct both normal and potentially faulty data. Anomalies are detected based on reconstruction error, assuming that faulty signals will have higher reconstruction errors due to deviation from learned normal patterns.

Model Architecture

The autoencoder is a simple feed-forward neural network consisting of an encoder, a latent bottleneck layer, and a decoder. All layers are fully connected (Dense) layers with ReLU activations, and the output layer uses linear activation to reconstruct the input signal. The model is trained using Mean Squared Error loss.

Training and Reconstruction

The model is trained only on normal signal windows. During inference, the model reconstructs the input signal, and the mean squared error (MSE) between the input and reconstruction is used as the anomaly score. A threshold, defined at the 40 percentiles of the MSE distribution on normal data, is used to classify samples as normal or anomalous.



The graph shows how the **Mean Squared Error (MSE)** loss changes over 50 training epochs:

- **Blue line**: Training loss
- **Orange line**: Validation loss
- Both losses decrease, which means the model is learning normal patterns
- The validation loss remains close to the training loss, indicating **good generalization** and **no significant overfitting**

This suggests the autoencoder has learned to reconstruct normal signals effectively.

Anomaly Detection and Evaluation

To evaluate the model's performance, a balanced subset of data (25,000 normal and 25,000 faulty samples) was used. Evaluation metrics include:
- Precision: Percentage of predicted anomalies that are truly faulty
- Recall: Percentage of actual faults correctly identified
- F1-score: Harmonic mean of precision and recall
- ROC Curve and AUC: Visual and numerical representation of model's classification performance
- Confusion Matrix: Summary of prediction results showing TP, FP, TN, FN

```
Loaded shape: (3531528, 100)
Label shape : (3531528,)
Unique labels: (array([0, 1]), array([1310400, 2221128]))

Anomaly detection threshold: 0.65157
Total Anomalies detected: 30000 out of 50000
```

```
 Classification Metrics
Precision : 0.6304 — % of predicted anomalies that are truly faulty
Recall    : 0.6014 — % of actual faults correctly identified
F1-Score  : 0.6155   — Harmonic mean of precision and recall
Accuracy  : 0.5275 — Overall correct predictions (both normal and faulty)

Detailed Classification Report:
              precision    recall  f1-score   support

           0     0.3732    0.4023    0.3872    262080
           1     0.6304    0.6014    0.6155    444225

    accuracy                         0.5275    706305
   macro avg     0.5018    0.5019    0.5014    706305
weighted avg     0.5350    0.5275    0.5308    706305
```

The autoencoder model was tested on over **3.5 million signal windows**, each with a size of 100. The labels represent:

- **0** = normal
- **1** = faulty

Using a reconstruction error threshold of **0.65157**, the model flagged **30,000 anomalies out of 50,000 predictions**.

Main Results:

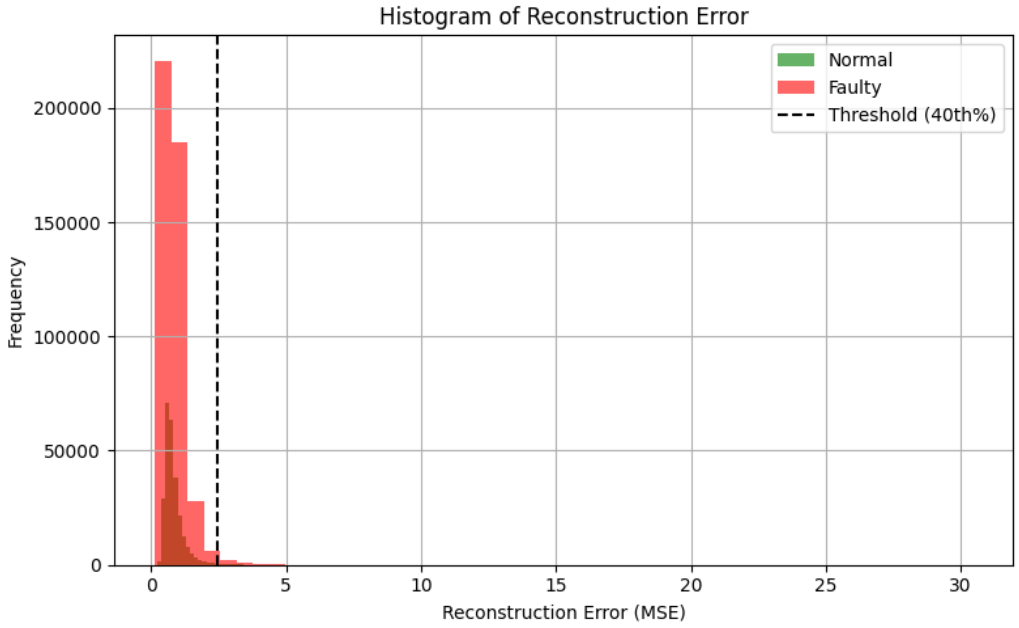
- **Precision = 63.04%** → Most of the predicted anomalies were correct
- **Recall = 60.14%** → It correctly found 60% of the real faults
- **F1-Score = 61.55%** → Balanced score between precision and recall
- **Accuracy = 52.75%** → Overall correct predictions (normal + faulty)

These results show that the model can detect many faulty signals, but still misclassifies some normal ones.

Visualizations

Several visualization techniques were used to enhance interpretability:

- Histogram of Reconstruction Error: Shows natural separation between normal and faulty samples



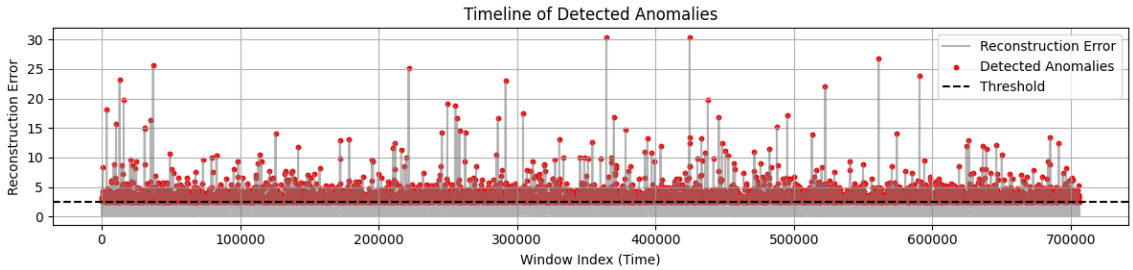
The histogram above compares the reconstruction error distributions of normal and faulty samples based on Mean Squared Error (MSE) computed by the autoencoder.

- **Green bars** represent normal samples
- **Red bars** represent faulty samples
- The **dashed vertical line** marks the anomaly detection threshold (set at the 40th percentile)

This plot reveals that while both normal and faulty data have overlapping error distributions, faulty samples show a **longer right tail**, indicating more frequent high-error reconstructions. This pattern supports the use of reconstruction error as a signal for anomaly detection, although the overlap suggests a trade-off between **false positives** and **false negatives**, depending on the chosen threshold.

The histogram is instrumental in justifying threshold selection and evaluating model sensitivity.

- **Timeline of Detected Anomalies**

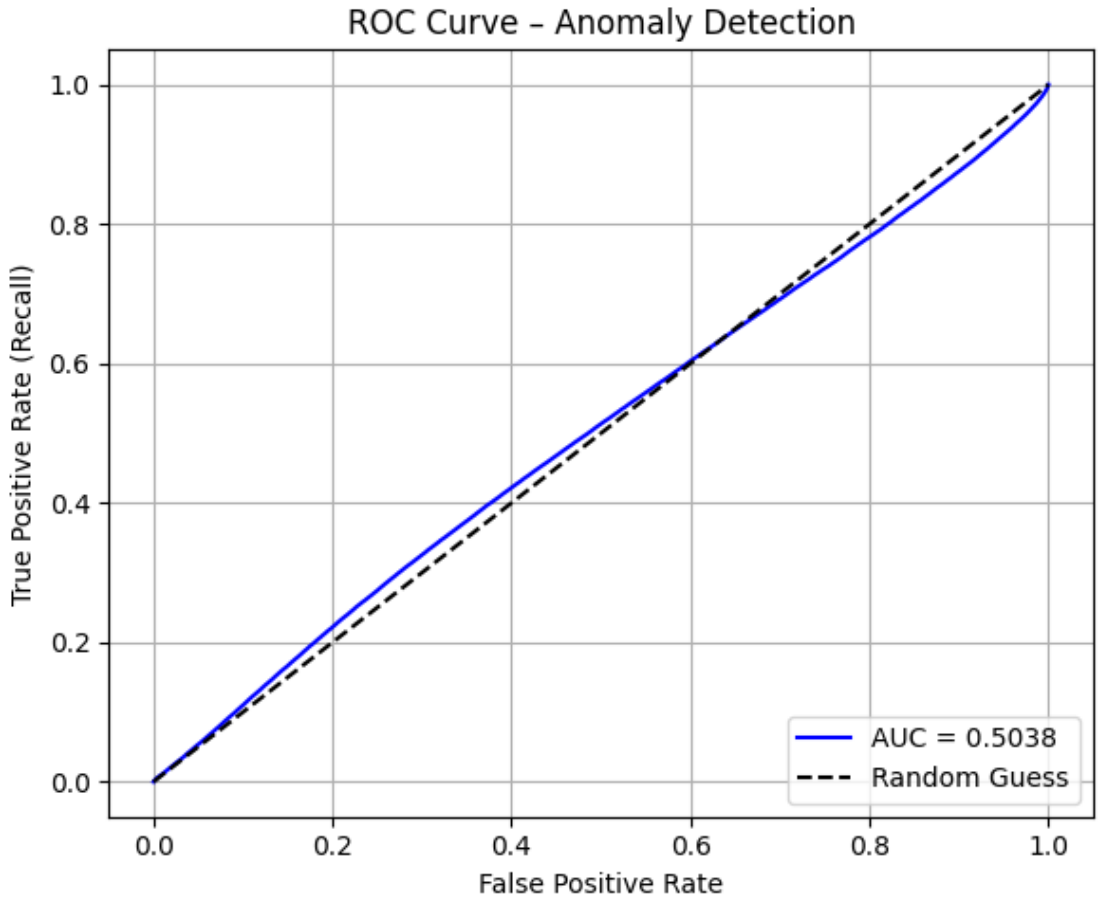


The above plot illustrates the timeline of detected anomalies based on the reconstruction error produced by the autoencoder model:

- The **X-axis** represents the index of the sliding windows, corresponding to time.
- The **Y-axis** shows the reconstruction error for each windowed input.
- The **gray line** represents the raw reconstruction error trend.
- The **dashed black line** is the predefined threshold, above which samples are considered anomalous.
- **Red dots** indicate individual windows where the reconstruction error exceeds the threshold, and thus an anomaly is flagged.

This timeline visualization is essential for diagnosing **when** faults begin to manifest in the machine's behavior. The model successfully highlights intermittent spikes that deviate significantly from normal patterns, making it a powerful tool for predictive maintenance and early fault detection in industrial systems.

- **ROC Curve**



The Receiver Operating Characteristic (ROC) curve provides insight into the model's ability to distinguish between normal and anomalous samples across various threshold settings. The curve plots the **True Positive Rate (Recall)** against the **False Positive Rate**.

In this case, the **Area Under the Curve (AUC)** is **0.5038**, which is only slightly better than random guessing (AUC = 0.5). This indicates that the model currently has **very limited discriminative power** and is not effectively separating faulty behavior from normal operation.

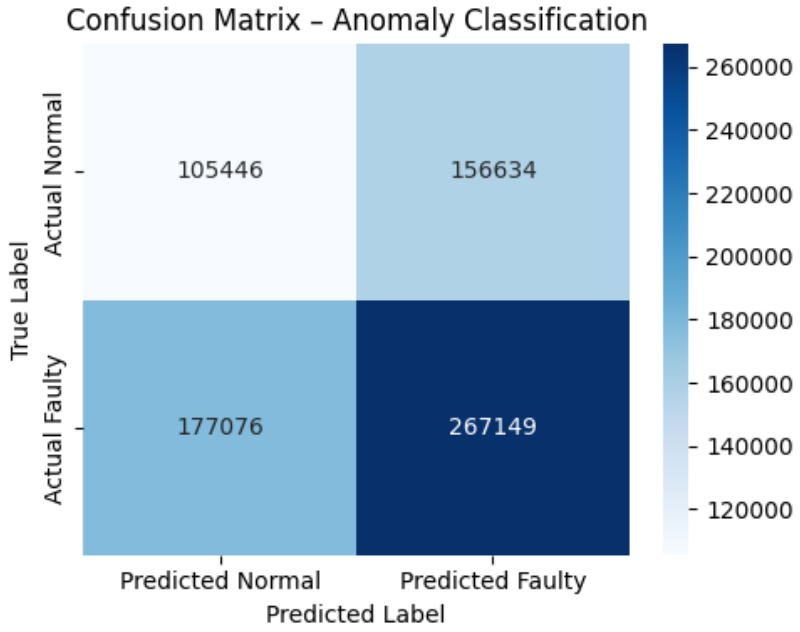
This result suggests the need for:

- Further model refinement (e.g., improved architecture or input features)
- Better threshold selection or anomaly scoring techniques
- Enhanced preprocessing or alternative embeddings

Improving the model's ability to generate distinct reconstruction errors for anomalies vs normal data may help increase the AUC and make the detector more useful in practical applications.

- **Confusion Matrix**

The confusion matrix above summarizes the performance of the anomaly detection system in terms of predicted vs actual labels:

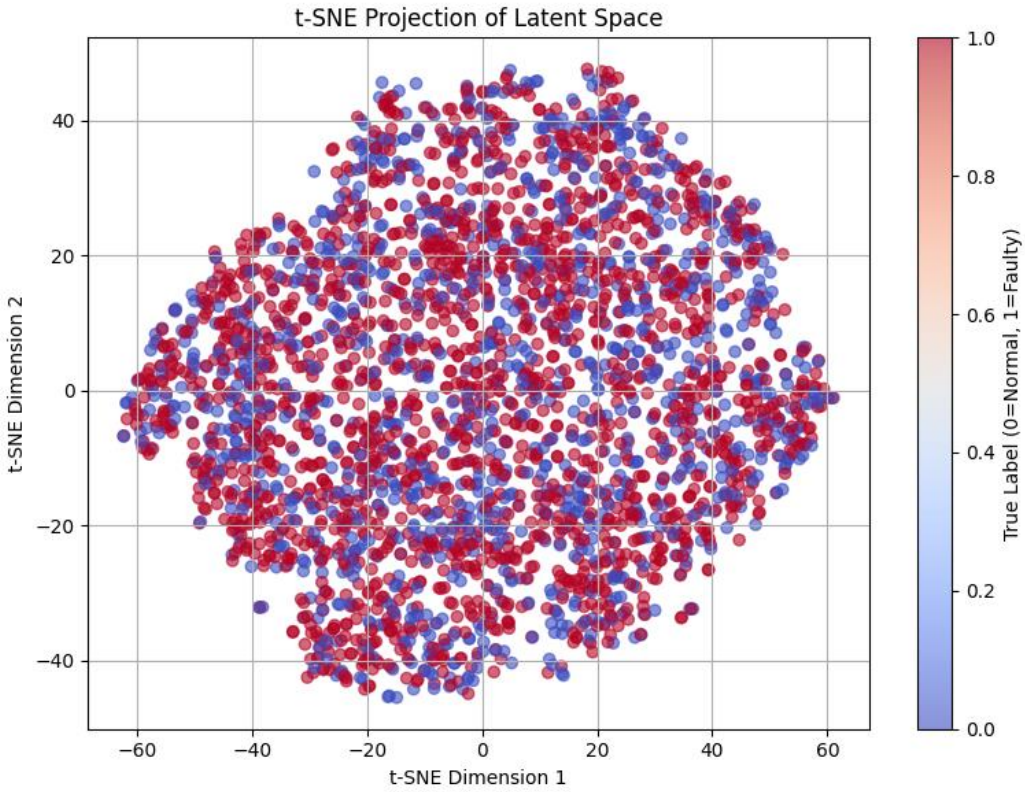



```
 Confusion Matrix Breakdown:
True Negatives : 105446  – Correctly predicted normal samples
False Positives: 156634  – Normal samples incorrectly flagged as anomalies
False Negatives: 177076  – Faulty samples missed by the model
True Positives : 267149  – Correctly predicted faulty samples
```

- **True Positives (TP):** 267,149 — Faulty samples correctly identified as anomalies
- **True Negatives (TN):** 105,446 — Normal samples correctly identified as non-anomalous
- **False Positives (FP):** 156,634 — Normal samples incorrectly flagged as anomalies
- **False Negatives (FN):** 177,076 — Faulty samples incorrectly classified as normal

The results indicate that the model performs well in identifying a large number of anomalies but also produces a notable number of both false positives and false negatives. This suggests that while the autoencoder is sensitive to changes in signal behavior, further optimization (e.g., threshold tuning or post-filtering) may be needed to improve precision and recall for practical deployment.

- **Latent Space Visualization (t-SNE)**



To better understand how the autoencoder internally represents the input signals, we applied t-SNE to project the high-dimensional latent vectors into two dimensions. Each point in the plot corresponds to a signal window, and the color indicates the true label:

- **Blue (0)**: Normal samples
- **Red (1)**: Faulty samples

Although there is some clustering of faulty and normal data, the overlap between the two classes indicates that anomaly separation is not perfectly linear in the latent space. This reflects the real-world complexity of fault patterns and highlights the need for more robust thresholds or additional modeling techniques for accurate anomaly detection.

Multimodal LLM Integration

To improve interpretability, I tried to integrate a multimodal LLM (e.g., BLIP-2) to generate natural language explanations from visual anomaly plots. By passing images like

reconstruction error timelines to the model, we can obtain intuitive descriptions of anomalies. This feature makes the system more accessible to non-technical users and supports human-readable reporting.

Challenges Faced

- Data Imbalance: The original dataset had a huge number of normal samples and very few faulty ones, which made evaluation tricky. I had to create a synthetic balance for fair testing.
- Latency: Using tools like t-SNE and BLIP-2 on Google Colab caused delays due to high memory and compute demands.
- Session Crashes: Some notebook sessions crashed, especially when training models or running heavy visualizations.
- Interpretability: Understanding and trusting the results wasn't easy—so I had to rely on multiple visualizations like reconstruction errors, ROC curves, and confusion matrices.
- Thresholding: Setting the anomaly detection threshold wasn't straightforward; I had to try different percentiles and observe the impact.

Future Work and MLOps Integration

Planned future extensions include:
- Streamlit UI for real-time data monitoring and visualization
- Experiment Tracking using MLflow or TensorBoard
- Docker-based deployment for consistent environment management
- REST API using FastAPI or Flask for serving predictions
- Monitoring and automatic retraining pipeline for adapting to data drift

Conclusion

In this project, I implemented an unsupervised anomaly detection pipeline using a dense autoencoder to model normal operational behavior of machinery based on the NASA bearing dataset. By leveraging reconstruction error as an indicator of abnormality, the system was able to detect deviations from learned patterns without relying on labeled data during training.

A balanced evaluation set was created using stratified sampling to fairly assess model performance through metrics such as precision, recall, F1-score, ROC AUC, and confusion matrix breakdown. While the recall remained relatively low, indicating difficulty in

detecting all faulty samples, the results reflect the challenges of real-world imbalanced datasets and the inherent limitations of using a simple thresholding technique.

The project demonstrates the potential of autoencoders for predictive maintenance in industrial IoT systems. With further refinement, including threshold tuning, temporal smoothing, and integration of additional sensor features, this system could serve as a building block for real-time fault detection. Future work may also involve deploying the solution using Streamlit UI, Docker, or REST API for practical usability and integration into operational environments.