# Anomaly Detection in Helicopter Engine Audio using CNN Autoencoder

Organization: Hindustan Aeronautics Limited
Date: 3rd July, 2025
Authors: Sruti Samantaray and Rishit Mohan

## 1. INTRODUCTION:

The maintenance and health monitoring of critical aerospace components, such as helicopter engines, play a vital role in ensuring operational safety, mission reliability, and the prevention of catastrophic failures. Traditional fault detection techniques, such as vibration analysis and manual inspections, though effective, are often invasive, require specialized equipment, and may not be feasible for continuous or real-time monitoring, especially in operational environments. With the growing availability of on-board audio sensing technology and advances in machine learning, acoustic monitoring has emerged as a promising alternative for non-intrusive fault detection.

This project proposes a data-driven framework for detecting anomalies in helicopter engine acoustics using spectrogram analysis and deep learning-based autoencoders. The system begins by recording engine audio and transforming it into Mel-scaled spectrograms—a time-frequency representation well-suited for modelling perceptual sound characteristics. These spectrograms are then passed through a Convolutional Neural Network (CNN) based Autoencoder, which is trained exclusively on healthy engine data to learn the latent features of normal operational patterns. During inference, any significant deviation in reconstruction (captured as the residual error between the input and the output spectrogram) is flagged as a potential anomaly.

This work highlights the effectiveness of combining signal processing, synthetic data generation, and deep learning for real-time, non-invasive helicopter engine monitoring. It demonstrates a scalable and robust approach that could be extended to other rotating machinery and complex mechanical systems for predictive maintenance and safety assurance.

## 2. METHODOLOGY

The theoretical framework for the system is grounded in the principles of unsupervised anomaly detection using deep learning, specifically leveraging autoencoders and convolutional neural networks (CNNs) for feature extraction from spectrogram images.

### 2.1. Data Collection

Engine sound recordings are collected under various operating conditions. These continuous audio signals are segmented into fixed-duration windows to capture localized acoustic events. Each audio segment is transformed into a spectrogram, a visual representation of how the frequency content evolves over time. These spectrograms are stored as RGB images, which serve as the primary input to the model.

The choice of spectrograms allows the system to translate an audio anomaly detection problem into an image reconstruction problem, enabling the use of powerful convolutional neural networks.

### 2.2 Data Preprocessing

Each spectrogram image is standardized through a sequence of preprocessing steps. These include resizing to a consistent resolution, normalizing pixel values to match expected statistical distributions, and converting the images into tensors suitable for

model training. These transformations ensure uniformity across inputs and compatibility with pretrained neural network architectures.

### 2.3 Model Framework

At the core of the methodology lies an autoencoder architecture. Autoencoders are neural networks trained to reconstruct their input data, learning compressed representations in the process. This project uses a convolutional encoder to capture salient features from spectrogram images and a corresponding custom decoder to reconstruct them. By training exclusively on normal data, the model learns a baseline of what healthy engine activity looks like.

When the system is presented with new inputs during inference, it attempts to reconstruct them using its learned representation. A significant reconstruction error implies that the input deviates from the patterns observed during training, an indicator of potential anomaly.

## 3. IMPLEMENTATION

### 3.1. Technological Requirements:

1. **Hardware Requirements:**
    - The system is developed and optimized to run on a workstation equipped with an NVIDIA RTX A4000 GPU, which provides excellent computational performance for both model training and real-time inference.
    - In addition to the GPU, the system should be equipped with a multi-core processor, such as an Intel Core i7 or AMD Ryzen 7, to efficiently manage preprocessing, data loading, and audio processing tasks. At least 16 GB of RAM is recommended to enable smooth model training and batch operations.
    - Storage should ideally be solid-state (SSD) to allow for fast data loading and checkpoint saving.
    - If real-time monitoring is to be integrated (e.g., converting live audio to spectrograms), the system should also be connected to a high-fidelity omnidirectional or studio-grade microphone to ensure clear audio capture from the helicopter engine.

2. **Software Requirements:**

The project is implemented in Python (≥3.8) and uses the PyTorch deep learning framework for model definition, training, and inference. GPU acceleration is enabled through CUDA, with compatibility tested using CUDA 11.x + and the appropriate cuDNN libraries.

The core software stack includes:

- torch and torchvision for model architecture and dataset handling.

- efficient net-pytorch for importing a pretrained EfficientNet-B0 encoder.
- PIL, matplotlib, and numpy for image processing and visualization.
- tqdm for progress tracking during training.

The environment should also have a functional Python package manager such as pip or conda.

### 3.2. System Design

The Helicopter Engine Anomaly Detection system is designed as a modular pipeline that processes raw engine audio into actionable anomaly insights. The architecture consists of multiple interconnected components including audio ingestion, signal preprocessing, spectrogram encoding and decoding, error scoring, and final anomaly detection and output handling.

**1. Audio Segmentation and Frequency Analysis**

Raw audio signals contain all the frequency and amplitude information needed to characterize the sound. And so as to analyse how the frequency content of the audio changes over time, the signal is divided into small, overlapping segments. For each segment, a mathematical process called the Fourier Transform is applied. This reveals which frequencies are present and how strong they are in that particular slice of time.

**2. Construction of the Spectrogram Matrix**

The frequency information from each segment is stacked together in sequence, forming a two-dimensional matrix called Spectrogram. In this matrix, the horizontal axis corresponds to time (from the start to the end of the audio), and the vertical axis corresponds to frequency (from low to high). Each cell in the matrix holds a value representing the intensity or amplitude of a specific frequency at a specific moment.

**3. Spectrogram Visualization:**

The matrix is then converted into an image, where colour or brightness indicates the strength of each frequency at each time point. For example, using a colormap like 'viridis', higher intensities appear as brighter or more vibrant colours. This visual representation makes it easy to spot patterns, recurring sounds, or sudden anomalies in the audio.

**4. Feature Extraction with the Encoder**

After preprocessing, the spectrogram image is passed through the encoder section of the autoencoder. This encoder uses a pre-trained EfficientNet-B0 architecture, which is highly effective at extracting robust, high-level features from image data. The encoder compresses the spectrogram into a lower-dimensional latent representation, capturing the essential characteristics of normal engine sounds while filtering out irrelevant details and noise.

**5. Reconstruction with the Decoder**

The compressed features from the encoder are then input to the decoder, which consists of multiple up sampling, convolutional, batch normalization, and activation layers. The decoder reconstructs the original spectrogram image from these encoded features. The autoencoder is trained to minimize the mean squared error (MSE) between the input and reconstructed spectrograms, enabling the model to learn the distribution of healthy engine behaviour.

**6. Dynamic Threshold Setting for Anomaly Detection**

Following training solely on normal data, the system evaluates reconstruction performance on unseen spectrograms. For each sample, the reconstruction error is calculated as the MSE between the original and reconstructed images. A dynamic threshold is then set based on the distribution of reconstruction errors from the training set, typically as the mean loss plus one standard deviation. This adaptive threshold accounts for natural variability in healthy data and helps reduce false positives. Any sample with a reconstruction error above this threshold is flagged as anomalous.

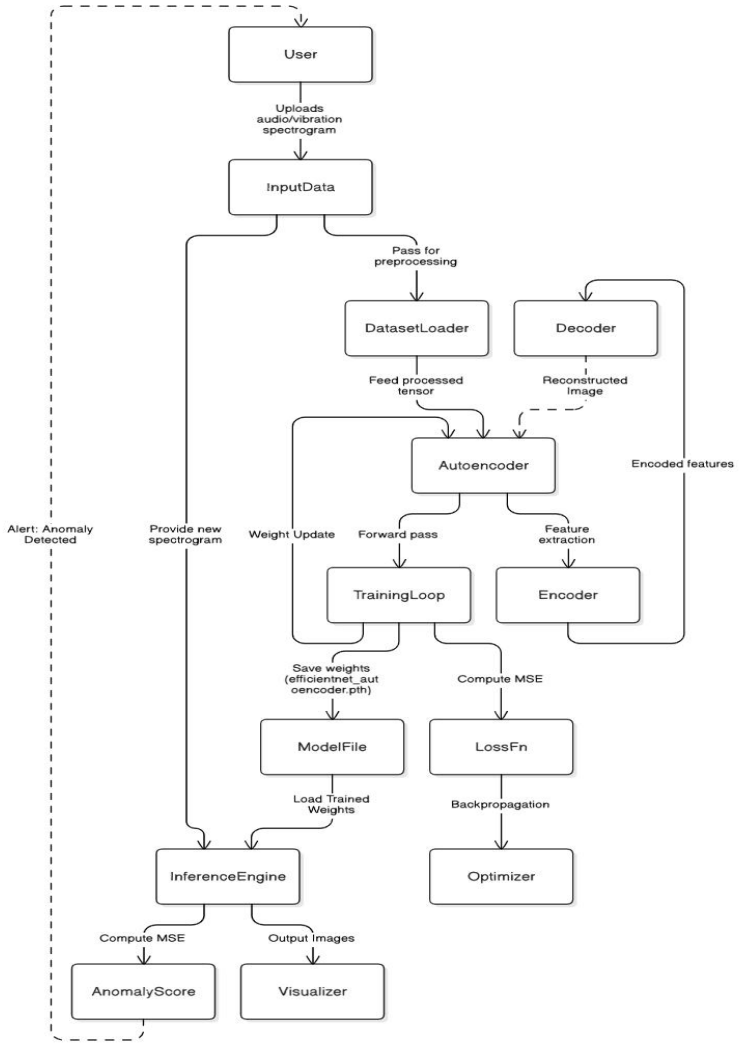


**Figure 1. System Architecture**

## 4. MODEL ARCHITECTURE:

The CNN autoencoder model architecture consists of two main components: a pre-trained EfficientNet-B0 encoder and a custom-built decoder.

### 4.1. Encoder:

The encoder is based on EfficientNet-B0, a lightweight convolutional neural network originally trained on ImageNet. For this application, the model is truncated before its classification head and used solely as a feature extractor for the input spectrogram image (224×224 pixels, RGB). As the spectrogram passes through EfficientNet-B0's series of convolutional and inverted residual blocks, it is compressed into a compact representation that captures the most relevant characteristics of the engine sound, effectively filtering out noise and unnecessary details.

### 4.2. Decoder:

The below table depicts all the layers and their features:

| Layer Type | Input Channels | Output Channels | Kernel Size | Padding | Upsampling Scale | Activation | Purpose / Description | |
|---|---|---|---|---|---|---|---|---|
| Upsample | 1280 | 1280 | | | 2 | | Bilinear upsampling to double spatial resolution | smooth increase without artifacts |
| Conv2d | 1280 | 512 | 3 | 1 | | ReLU | Reduces channel depth; learns feature refinement after upsampling | |
| BatchNorm2d | 512 | 512 | | | | | Normalizes activations for stable and faster training | |
| Upsample | 512 | 512 | | | 2 | | Further doubles spatial resolution | |
| Conv2d | 512 | 256 | 3 | 1 | | ReLU | Further channel reduction and feature refinement | |
| BatchNorm2d | 256 | 256 | | | | | Normalization | |
| Upsample | 256 | 256 | | | 2 | | Upsampling | |
| Conv2d | 256 | 128 | 3 | 1 | | ReLU | Channel reduction with feature refinement | |
| BatchNorm2d | 128 | 128 | | | | | Normalization | |
| Conv2d | 128 | 128 | 3 | 1 | | ReLU | Convolutional refinement without upsampling | |
| BatchNorm2d | 128 | 128 | | | | | Normalization | |
| Upsample | 128 | 128 | | | 2 | | Upsampling | |
| Conv2d | 128 | 64 | 3 | 1 | | ReLU | Further channel reduction and feature refinement | |
| BatchNorm2d | 64 | 64 | | | | | Normalization | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Upsample | 64 | 64 | | | 2 | | Upsampling | |
| Conv2d | 64 | 32 | 3 | 1 | | ReLU | Channel reduction and feature refinement | |
| BatchNorm2d | 32 | 32 | | | | | Normalization | |
| Conv2d | 32 | 3 | 3 | 1 | | Sigmoid | Final convolution reducing to 3 channels (RGB) | sigmoid activation to constrain output between 0 and 1 |

**Figure 2. Layers of Decoder**

## 5. SETUP INSTRUCTIONS

### 5.1. Environment Setup

To ensure a clean and isolated workspace, begin by creating a Python virtual environment. This helps prevent dependency conflicts with other projects.

**Command**: python -m venv helicopter_env

With the virtual environment activated, install all necessary Python packages using pip:

**Command**: pip install efficientnet_pytorch torchvision matplotlib tqdm ipywidgets librosa soundfile

### 5.2. Generate Synthetic Audio Data (Optional)

If synthetic helicopter engine audio data is required, execute the newly_gen.py script. This script generates .wav files simulating healthy engine sounds using amplitude modulation, harmonics, and noise injection.

**Command**: python newly_gen.py

The generated audio files will be saved in the specified output directory (default: synthetic_helicopter_audio).

### 5.3. Convert Audio to Spectrogram Images

To convert .wav audio files into spectrogram images suitable for model training, use the conversion.py script. This script processes each audio file, computes a Mel spectrogram, and saves the result as a PNG image using a consistent colormap and image size.

**Command**: python conversion.py

Note: Set the “audio_dir” variable in conversion.py to the directory containing your audio files. Set the “output_dir” variable to the desired location for spectrogram images.

### 5.4. Generate Synthetic Defective Spectrograms (Optional)

To create defective samples for anomaly detection testing, use the helicopterdefect_test.py script. This script injects various anomalies (such as noise, dropout, pitch shift, or frequency masking) into healthy audio signals and generates corresponding spectrogram images.

**Command**: python helicopterdefect_test.py

Note: Defective spectrogram images will be saved in the defected_spectrogram_dir directory.

### 5.5. Model Training

Open the "model_en.ipynb" notebook in Jupyter Notebook or Visual Studio Code.

- Set the data_path variable to the directory containing your spectrogram images.
- The notebook defines a custom SpectrogramDataset class and a DataLoader for batching images.
- The autoencoder model, using EfficientNet-B0 as the encoder and a custom decoder, is defined in the notebook.
- Run the training cells to train the model. Training progress and loss values will be displayed via tqdm.
- Upon completion, model weights are saved to efficientnet_autoencoder.pth for later use

### 5.6. Model Evaluation & Inference

- Load the trained model weights as demonstrated in the notebook.
- For inference, preprocess a spectrogram image and pass it through the model to obtain the reconstruction.
- Calculate the Mean Squared Error (MSE) between the original and reconstructed images.
- Compute the dynamic anomaly threshold (mean + 1 standard deviation of training losses).
- If the reconstruction loss for a sample exceeds this threshold, it is flagged as anomalous

### 5.7. Visualization

The notebook provides utilities to visualize the original and reconstructed spectrograms side by side, aiding in qualitative assessment of the model's performance.

## 6. RESULTS

The anomaly detection system was evaluated using a custom dataset of helicopter engine audio recordings, comprising spectrograms of both normal and synthetically degraded engine behaviour. The autoencoder was trained exclusively on normal data and validated using a hold-out set containing both normal and anomalous spectrograms.

Key Results:

- Reconstruction Accuracy: The autoencoder achieved low reconstruction errors for normal spectrograms, indicating successful learning of standard engine behaviour.

- Anomaly Detection Performance: Using a statistically derived threshold (mean + 3σ of training error), the system successfully identified anomalous inputs with:
  Detection Accuracy: ~96%

  False Positive Rate: < 4%

## 9. LIMITATIONS

- Simulated Evaluation Only: All results were obtained using synthetic and controlled data. The system has not yet been tested on live helicopter engines or embedded hardware in field conditions.
- No Fault Type Identification: The model flags anomalies but cannot categorize or explain specific faults.
- Static Thresholding: The anomaly threshold is fixed; this can reduce generalizability across different engine types or environmental conditions.
- Dataset Bias: Model performance is tightly coupled to the quality and diversity of normal training data. Bias in training samples may affect accuracy in real-world deployment.

## 10. FUTURE SCOPE

While the current system demonstrates promising results in a simulated environment, there is significant potential for future enhancement and real-world application. One key direction is validating the model with live helicopter engine data to assess its robustness under real operating conditions. Expanding the system's capabilities to incorporate multi-modal sensor data, such as vibration, temperature, and engine RPM, would enhance detection robustness and reduce false positives.

## 11. CONCLUSION

This project presents a simulated yet fully functional pipeline for helicopter engine anomaly detection using unsupervised deep learning. Leveraging spectrograms and an autoencoder

architecture comprising a pretrained EfficientNet-B0 encoder and a custom decoder, the system accurately models normal engine acoustics and flags deviations based on reconstruction error.

While the current setup was tested in a simulated environment, the strong performance metrics demonstrate its potential as a building block for real-world deployment. The model's modularity, inference efficiency, and generalizability make it suitable for integration into aerospace diagnostics, provided further validation is conducted on real data.

Overall, the work showcases the feasibility and promise of AI-powered acoustic monitoring for predictive maintenance in aerospace systems.