Anomaly Detection - Challenge 2 AML

Alex Argese
Student
EURECOM
Biot, France
alex.argese@eurecom.fr

Cristian Degni
Student
EURECOM
Biot, France
cristian.degni@eurecom.fr

Miriam Lamari
Student
EURECOM
Biot, France
miriam.lamari@eurecom.fr

Enrico Sbuttoni
Student
EURECOM
Biot, France
enrico.sbuttoni@eurecom.fr

***Abstract*—This report presents a comparative analysis of unsupervised machine learning techniques for detecting anomalous sounds in industrial equipment, specifically focusing on the slider machine type. The primary challenge addressed is identifying unknown anomalous sounds in test samples, given only normal sound recordings during training. We describe the dataset, which includes audio recordings from the ToyADMOS and MIMII collections, and detail the preprocessing steps such as audio normalization, feature extraction using log-mel spectrograms. Various models were evaluated, including standard Autoencoders, Variational Autoencoders (VAEs), Convolutional Autoencoders (CAEs), and the PANNs (Pretrained Audio Neural Networks) inference framework. The models were assessed using Area Under the Curve (AUC) as the primary metric to reflect their ability to rank samples by anomaly likelihood. Results highlight the strengths and limitations of reconstruction-based methods versus embedding-based representations, with PANNs showing promising generalization across machine IDs. This work underscores the potential of deep unsupervised methods for robust anomaly detection in real-world, noise-contaminated industrial environments.**

***Index Terms*—Machine Learning, Anomaly Detection, Sound processing, Autoencoder, Pannes Inference**

I. Introduction

Detection of **anomalies** in **industrial equipment** is a critical task for ensuring **operational efficiency** and preventing costly **downtime**. In this challenge, we focus on detecting **anomalous sounds** from a specific type of machine, the **slider**, using **unsupervised machine learning** techniques. The primary goal is to identify **unknown anomalous sounds** in test samples, given only **normal sound recordings** during training. The dataset used for this task consists of audio recordings from the **MIMII** dataset.

II. Dataset

The dataset consists of **audio recordings** in **.wav** format, collected with **eight microphones** placed around a **slider machine**. All recordings are treated as a **single-channel** input from a **fixed microphone**. Each recording is approximately **10 seconds** long and includes both the **operating sound** of the slider machine and **environmental noise**. All signals are **downsampled** to a **sampling rate** of **16 kHz**.

The **training set** contains only **normal sound recordings**, while the **test set** includes both **normal** and **anomalous sounds**.

The dataset is organized as follows:

- **dev/ folder**: contains around **1,000 normal samples** for training and **100–200** samples each of **normal** and **anomalous sounds** for testing.
- **eval/ folder**: includes approximately **400 unlabeled test samples** for evaluation, along with additional **normal samples** that may be used for training.

The **class distribution** within the test set is **unbalanced**:

- **Class Anomalous**: 801 sounds
- **Class Normal**: 300 sounds

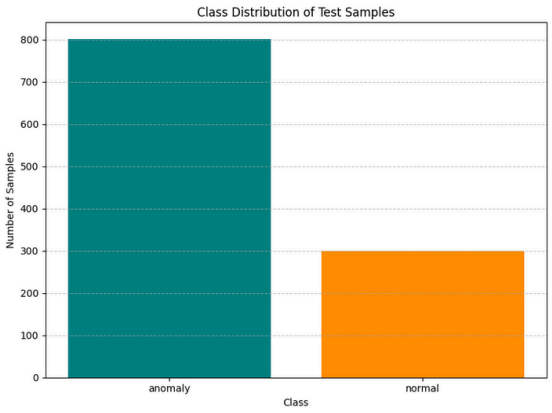

Fig. 1: Test samples class distribution.

III. Preprocessing

A. Mel-spectrogram transformation

Conversion of **waveform** signals into a **mel-spectrogram** representation is a common practice in **audio analysis**, especially suitable for **machine listening** tasks such as **anomaly detection**.

The preprocessing pipeline involves first **loading** the **.wav** audio files using the **Librosa** Python library, followed by

mel-spectrogram extraction using the following parameters:

- **number of FFT components**: defines the number of samples used in each Fourier Transform window;
- **hop length**: determines the step between successive analysis frames;
- **number of mel bands**: sets the resolution of the mel frequency axis.

The resulting **mel-spectrograms** are converted to the **decibel scale** and then **reshaped into flattened vectors** for efficient storage and compatibility with model input formats. The extracted features are **organized into separate folders** for training and testing purposes.

A **mel-spectrogram** looks like this:

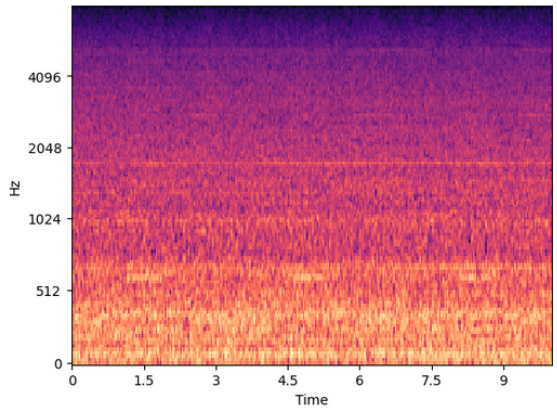

Fig. 2: Example of a mel-spectrogram.

B. Feature Normalization

After converting the **audio signals** into **mel-spectrogram** representations and flattening them into **1D feature vectors**, we applied **feature normalization** to ensure **numerical stability** and improve **model performance**.

We used **z-score standardization**, which transforms each feature to have a **mean of 0** and a **standard deviation of 1**. This is particularly important for models that are **sensitive to feature magnitudes**, such as **support vector machines** and **neural networks**. Without normalization, features with **larger numerical ranges** could **dominate the learning process** or **slow convergence**.

The **normalization parameters** (mean and standard deviation) were computed using the entire set of **normal training samples**. This is consistent with the **anomaly detection** framework, where it is assumed that only **normal examples** are available during training.

IV. Models Evaluated

A. Fully-Connected Autoencoder

A **fully-connected Autoencoder** was used to learn **compressed representations** from **tabular input features**. Implemented in **PyTorch**, the model consists of a **dense encoder** that maps input vectors to a **latent space** of dimension **64**, followed by a **symmetric decoder** that reconstructs the original input.

The model is trained using **mean squared error (MSE)** as the reconstruction loss and optimized with the **Adam optimizer** over **30 epochs**. A **learning rate scheduler** is applied to adjust the learning rate dynamically and help prevent overfitting.

TABLE I: Hyperparameter setup for Fully-Connected Autoencoder

Hyperparameter	Ranges / Values	Used
latent_dim	64 (AE), 8 (VAE)	64 / 8
optimizer	Adam	Adam
lr	1e-3	1e-3
β	0.001	0.001

B. Convolutional Variational Autoencoder

A **Convolutional Variational Autoencoder (ConvVAE)** was used to learn **low-dimensional embeddings** of **Mel spectrograms** derived from **16kHz audio data**. The model, implemented in **PyTorch**, includes a **convolutional encoder** that projects inputs to a **latent space** of dimension **8**, **16**, or **32**, and a **decoder** that reconstructs the spectrogram from the latent representation. Input spectrograms are **standardized** and **padded** to maintain spatial compatibility throughout the network.

The architecture applies the **reparameterization trick** to enable stochastic sampling during training and combines **reconstruction loss** with **KL divergence** using a **β-VAE loss formulation**. The model supports configurable **latent dimensionality** and **β weighting**, allowing tuning of the trade-off between reconstruction accuracy and latent space regularization.

TABLE II: Hyperparameter setup for ConvVAE

Hyperparameter	Ranges / Values	Used
latent_dim	8, 16, 32	8
β	0.01, 0.1, 1.0	0.1

C. Variational Autoencoder

A **Variational Autoencoder (VAE)** was used to learn **compact latent representations** from tabular data. The model is implemented in **PyTorch** and features a **fully-connected encoder** that maps inputs to a **latent space** of dimension **8**, using separate layers to estimate the **mean** and **log-variance**.

A **decoder** reconstructs the input from a latent variable obtained through the **reparameterization trick**. The loss function combines **mean squared error (MSE)** for reconstruction and **KL divergence** for regularization, forming a **β-VAE loss**. The model is trained with the **Adam optimizer**, and a **learning rate scheduler** is used to reduce learning rate when performance plateaus.

TABLE III: Hyperparameter setup for VAE

Hyperparameter	Ranges / Values	Used
latent_dim	8, 16	8
optimizer	Adam	Adam
lr	1e-3	1e-3
β	0.001	0.001

D. PANNs Pretrained Model

The model **Cnn14** is a **pretrained convolutional neural network** from the **PANNs** (Pretrained Audio Neural Networks) family. It was trained on the **AudioSet** dataset and is designed for **general-purpose audio tagging**. The model takes **raw waveform input** and produces a fixed-size **2048-dimensional embedding** that captures **high-level acoustic features**.

Implemented via the **panns_inference** library, the model is used in **inference mode only**, with no additional training or fine-tuning. Audio inputs are **converted to mono**, **resampled to 32kHz**, and passed to the model through the provided **AudioTagging** interface. The resulting embeddings can be used for downstream tasks without modifying the network, leveraging its **pretrained audio representations**.

V. Metric Justification

Given the **class imbalance**, we used the **F1 score** as the main metric, as it balances **precision** and **recall** better than accuracy. This is especially important in **ecological monitoring**, where missing a cactus (**false negative**) is more critical than a false detection. F1 was therefore used for both **model selection** and **hyperparameter tuning**.

VI. Results Summary

A. Detailed Results – Logistic Regression

The best **logistic regression** model, trained with the selected **hyperparameters**, was evaluated on the **held-out test set**. The table below summarizes the **classification performance per class**:

The model achieved a final **test accuracy** of **81.01%**. It showed very high **precision** for class 0 (**no cactus**), but the relatively low **recall** (**0.55**) indicates many **false negatives**. For class 1 (**cactus**), the model achieved **strong performance** across all metrics, reflecting its **bias toward the majority class**. This suggests that while the **logistic regression model** benefits from the **augmentation strategy**, it still struggles with **class imbalance** and lacks the capacity to model **complex spatial patterns**.

B. Detailed Results – Support Vector Machine

The best-performing **SVM** model was evaluated on the **test set**. The performance breakdown per class is shown below:

The overall **test accuracy** was **95.8%**, with a **weighted F1 score** of **0.9582**. The SVM showed higher **recall** for the **cactus class** (class 1), and moderately improved **recall** on the **minority class** (class 0) compared to **logistic regression**. This confirms the model's ability to capture **non-linear decision boundaries** through the **RBF kernel**, although it still struggled with some **overlap in feature space**.

C. Detailed Results – Convolutional Neural Network

The best **CNN** model was evaluated on the **held-out test set**. The performance results per class are as follows:

The final **test accuracy** was **99.0%**, with a **weighted average F1 score** of **0.9896**. The **CNN** outperformed both **logistic regression** and **SVM** by a wide margin, particularly in identifying **minority class** (class 0) samples, thanks to its ability to learn **local spatial features** and **generalize well** from **augmented examples**.

D. Detailed Results – ResNet18

The **fine-tuned ResNet18** model achieved the **best performance** on the **test set**. Below is the detailed **classification report**:

ResNet18 reached a final **test accuracy** of 98.6%, with a **weighted F1 score** of 0.9857. It demonstrated **excellent generalization** and **balance between precision and recall** across both classes, validating the power of **transfer learning** even in **low-resolution**, **small-format image classification** tasks.

TABLE IV: Performance Comparison of Models on the Validation Set

Model	F1 Score	Accuracy
Logistic Regression	0.8098	81.01%
SVM	0.9582	95.82%
CNN	0.9896	98.96%
ResNet18	0.9857	98.57%

VII. Model selected

Although the **custom CNN** achieved a slightly higher **validation F1-score** (**0.9896**) than **ResNet18** (**0.9857**), we selected **ResNet18** as the **final model** due to its superior **robustness** and **generalization capabilities**. Its **pretrained layers** from **ImageNet** allow it to leverage **learned features** even on **small**, **low-resolution inputs** like our **32×32 aerial images**. This makes it more **reliable for deployment** on **unseen data**, where the **custom CNN** might be more prone to **overfitting**.

VIII. Inference on Unlabeled Test Set

After **model selection**, we applied the four **best-performing models** — **Logistic Regression**, **SVM**, **CNN**, and **ResNet18** — to the **4000 unlabeled images** from the **test set**. Each model was used to generate a **prediction (0 or 1)** for every image, maintaining the **order** of the images as read by the **DataLoader**.

To combine these predictions, we employed a **weighted majority voting** strategy, assigning a **normalized weight**

to each model based on its **accuracy** on the **internal test set**. The final class for each image was assigned as **1 (cactus)** if the **weighted sum** of predictions exceeded **0.5**, and **0** otherwise.

We opted for **weighted majority voting** instead of **simple majority voting** to give greater influence to **more accurate models**. This choice reflects our aim to **prioritize** the decisions of models that demonstrated **higher reliability** during **validation** and **internal test**, thus improving the **ensemble's overall robustness** and **predictive power**.

Finally, the resulting **predictions** were saved in a **CSV file** (ensemble_predictions.csv) with two columns: **id** (image filename) and **label** (predicted class).

IX. Conclusion and Next Steps

Despite its **simplicity**, **Logistic Regression** achieved a **strong baseline performance**, demonstrating that even **linear models** can be effective when supported by appropriate **preprocessing** and **balancing techniques**.

The **SVM** outperformed logistic regression, especially in terms of **class 1 recall**, but was still constrained by the lack of **spatial awareness** in **flattened input representations**.

The **custom CNN** significantly improved **classification accuracy** and **balance across classes**, confirming the advantage of **convolutional architectures** in **image-based ecological tasks**.

Among all tested models, **ResNet18** stood out with **outstanding precision**, **recall**, and **F1 scores**, confirming the effectiveness of **transfer learning** even when applied to **small aerial images** of **ecological relevance**.

X. References

This report is inspired by the VIGIA project as described in:

Efren López-Jiménez et al., **Columnar Cactus Recognition in Aerial Images using a Deep Learning Approach**, Ecological Informatics, 2019.