AML Challenge 2: Anomalous Sound Detection

Group 47

Jana JUROŠ Jana.Juros@eurecom.fr

Rea PINTAR Rea.Pintar@eurecom.fr

Ivona STOJANOVIĆ Ivona.Stojanovic@eurecom.fr

February 14, 2024

1 Problem setting

The main objective of this challenge is to use the sound data of various machines (toy car, toy conveyer, etc.) to predict if there are any problems with the object. Samples are grouped by types of the machines. We will be dealing only with the Slide rail machine type. In this challenge we are working with two datasets, the *Development* and the *Evaluation dataset*. The main premise is that our model(s) will be trained solely using 'normal' audio clips, i.e. sounds coming from machines that are working properly, but will be able to classify the sounds from the Test set which contains both normal and abnormal (*anomaly*) sounds. The abnormal/anomaly sounds indicate there's a problem with the machine, and we want our model to be able to detect that. It might sound abstract at first, but there are a number of ways to approach this problem, and in this challenge we will show some of them. Additionally, the two datasets we mentioned are used for different purposes. Firstly, we will try out a couple of different models, train them and test them all using the Development set, after which we will choose the best-performing model and re-train it using the Evaluation set. The submission file will be the predictions for the Evaluation set.

2 Chosen approach

We decided to use the given baseline as a foundation for our code and improve it, in order to get better results. The idea of the baseline was to use a **Variational Autoencoder (VAE)**, which is used to project the sound data into a latent space with the *encoder* and then reconstruct it with the *decoder*. This type of neural network works with images. This is why it is necessary to transform the sound data into images first. The images we are transforming the sound data into are called **mel-Spectrograms** and are widely used in the audio and signal processing domain. All of the needed steps to do this, along with a data loader is provided in the baseline notebook.

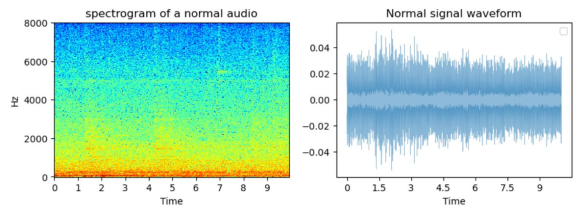
One of the ways we have tried to improve the baseline is batch normalization. **Batch normalization (B.N.)** is a technique commonly used in neural networks, including VAEs to improve training stability and performance. It normalizes the inputs of each layer by adjusting and scaling the activations using the mean and variance calculated over a mini-batch of training samples. This enables faster and more stable convergence during training. We considered this step necessary, as we had limited resources and tried to optimize the training process as much as we could. As we know, VAEs are extremely costly in terms of memory and training time.

As an additional model, we will try a different version of an Autoencoder, this time using a **Convolutional Normal Network** Autoencoder (CNNAE). The premise is the same as the VAE, but some of the principles differ. For example, with VAEs the latent space follows a specific probability distribution, often a multivariate Gaussian distribution, whereas with CNNAE it is typically a compact, dense vector that captures the essential features of the input data. It is not specifically designed to model a particular probability distribution. For this reason, the training process is much less costly both in terms of memory and time.

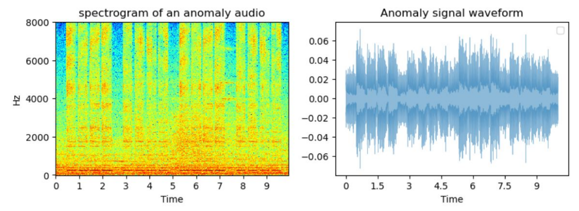
As a sidenote, we hereby mention that we have provided the code used for the implementation of VAE in our notebook, but opted to only run that code in the notebook locally, as we had extremely limited resources in terms of GPU space. In short, we were only able to run this code locally. For CNNAE this did not pose a problem, and the results of that model are shown in our notebook as it ran on Kaggle without issue.

Finally, the submission will be done using the following principle. We will give our models (Autoencoders) the image of the (log) mel-Spectrogram, which they will try to replicate, after which an anomaly score will be calculated. This score signifies how much the original image and the replicated one differ, where a smaller anomaly score indicates higher similarity of the two images. These will, as we have mentioned, be stored in our submission file (for the test data of the Evaluation set). In reality, we could look at this as a classification problem and can combine the anomaly score with other characteristics of the given audio (such as peak amplitude, MFCC score[1] and others) in order to create a more complex (and hopefully more accurate) classification model.

3 Data analysis



(a) Normal Sound.



(b) Anomaly Sound.

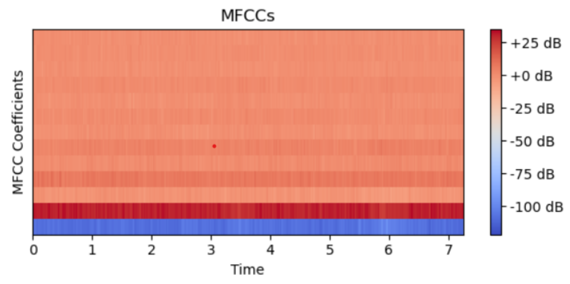
Figure 1: mel-Spectograms.

First, we will take a quick look at the raw datasets. Each of the audios are approx. 10 seconds long and include both the target machine's operating sound and environmental noise. The Development dataset contains 3471 audio samples from 3 different objects with the IDs `00`, `02` and `04`. There are 3204 sound files from three more objects with the IDs `01`, `03`, and `05` in the Evaluation dataset, on which we'll train (and test) the model we're going to submit. In both datasets, there is one machine that has only around 700 samples (machine `04` in the Development dataset and machine `05` in the Evaluation dataset). The other two machines have approximately double the number of files, with about 1400 each.

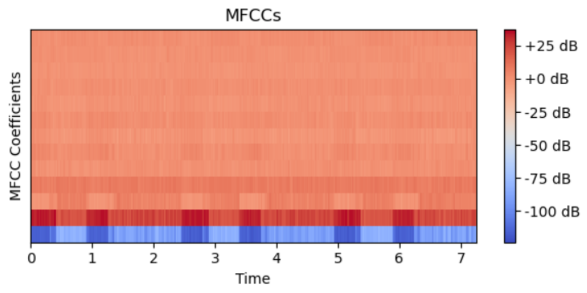
Let's familiarize ourselves with mel-Spectrograms (and log mel-Spectrograms, which is what we will give to our models for training and testing). Figure 1a represents the transformation of a normal sound audio clip from the time-frequency domain into the mel-Spectrogram.

In Figure 1b we can see how the Spectrogram will look different when we have an abnormal sound. This will ultimately be how we distinguish the two types of sounds.

What we will actually be giving to our models for training is a logarithm of the mel-Spectrogram, which closely resembles the original image in the sense that it shows the main patterns as they are in the original. The log-transformation is a commonly used technique in machine learning as the logarithm is a monotonically growing function, therefore preserving the relative ordering of values and not distorting results.



(a) Normal Sound.



(b) Anomaly Sound.

Figure 2: Mel-frequency Cepstral Coefficients displays.

We wanted to explore other sound representations but mel-Spectogram, so we also considered Mel-frequency Cepstral Coefficients (MFCCs), a commonly used representation of audio signals, often employed in speech and audio processing tasks. In Figure 2 we can see MFCC display of normal sound and anomaly sound. While the mel-Spectrogram provides a visual representation of the distribution of spectral energy over time, MFCCs offer a more compact representa-

[1]MFCC score - Mel Frequency Cepstral Coefficients, a widely used feature representation for audio signals

tion that captures the essential features of the audio signal, particularly the spectral shape and timbral characteristics. Additionally, MFCCs are derived from the mel-spectrogram by applying various transformations. Although the MFCC displays look promising, allowing us to visually clearly distinguish normal sounds from anomaly sounds, when we tried to use MFCCs with a CNN Autoencoder approach, we observed significantly worse results compared to mel-spectrograms. For this reason, we decided to stick with the log Mel-spectograms.

4 Model Selection

In this section we will present the neural networks that we chose to work with and describe their architecture in more detail.

4.1 Variational Autoencoder

Our implementation of a Variational Autoencoder is for the most part what was given in the baseline, with the exception that we added batch normalization between the linear layers of both the encoder and the decoder. The encoder network takes an input of dimension *x_dim* (640 in this case) and consists of three layers. The first layer is a linear layer that maps the input to a hidden dimension (*h_dim*) of 400. Batch normalization is applied after the first linear layer to normalize the outputs. The second layer is another linear layer that maps the hidden dimension to another hidden dimension, after which another batch normalization is applied. Finally, the third layer is a linear layer that maps the hidden dimension to a latent dimension (*z_dim*) of 20.

The decoder is a mirror image of the encoder, meant to reconstruct the input to the encoder. The first layer is a linear layer that maps the latent dimension back to the hidden dimension, followed by batch normalization. The second layer is another linear layer mapping the hidden dimension to another hidden dimension, again followed by batch normalization. The final layer is a linear layer that maps the hidden dimension to the output dimension (*x_dim*). Batch normalization is applied after this layer as well.

4.2 CNN Autoencoder

The encoder network in the CNNAE, the code for which was mostly taken from another baseline dealing with the same problem setting and dataset, has four convolutional layers. The first layer applies a 2D convolution with 32 filters, followed by a ReLU activation function. The second layer applies a 2D convolution with 64 filters, followed again by a ReLU activation function and a dropout layer with a dropout rate of 0.2. The third layer applies another 2D convolution with 128 filters, followed by ReLU and a dropout layer with a dropout rate of 0.3. The fourth layer applies a final 2D convolution with 256 filters, followed by ReLU and a dropout layer with a dropout rate of 0.3. Finally, the output is passed through a 2D convolution layer that reduces the dimensionality to the desired latent dimension (*z_dim*, which is 40 in this case). Once again, the decoder network is built to mirror the encoder, the output of the decoder being the reconstructed image of the encoder's input.

Optionally, the CNNAE supports skip connections between corresponding layers of the encoder and decoder. These skip connections are intended to enhance the reconstruction performance. If skip connections are enabled, the encoded feature maps from each layer of the encoder are concatenated with the corresponding decoded feature maps from the decoder before passing through the next layer. We will try out the model both with and without skip connections.

Lastly, for this model's training we will be using the SVDD (Support Vector Data Description) loss function. The SVDD loss function is used in one-class classification tasks to train an autoencoder-based model to learn a compact representation of normal data and identify anomalies. The goal is, in principle, to fit a hypersphere in the feature space that encapsulates the normal data points (normal sounds) while minimizing the volume of the hypersphere.

4.3 CNN Autoencoder with L2-Regularization

This model has the exact same network shape as the previous (CNNAE), with the addition of L2 regularization in the loss function. We implemented this by summing squared norms of all parameters of the model and multiplying the sum by the regularization strength (in our case 0.002), the result of which we added to the SVDD loss function. L2-regularization encourages the model's parameters to have smaller values, reducing the risk of overfitting.

5 Model Performance

The process of evaluating model performance will be conducted, as we have mentioned, using only the development set. As a metric, we will be using Area Under Curve (AUC) and Partial Area Under Curve (pAUC).

AUC is a popular metric often used in model performance evaluation, and pAUC is the variation of AUC where we have, other than the real and predicted values of the model, an additional parameter - maximum FPR (False Positive Rate). The metric pAUC represents the performance of the binary classifier in terms of discriminating between the positive and negative classes, considering the specified portion of the ROC curve. When we calculate the AUC (and pAUC), the real output of the model (*y_true*) is an array of zeroes, as y represents the error or difference of the original and reconstructed photo, which we want to be equal to 0. The predicted output, or rather, the actual difference of the model input and output is contained in the variable *y_pred*. Below we present the avergae AUC and pAUC scores (averaged on three different machines for which we have data in the development set) from which we will determine which model is better.

Model	AUC	pAUC
VAE	0.6881	0.5047
VAE with B.N.	0.8139	0.6412
CNNAE	0.8902	0.7294
CNNAE with L2	**0.9045**	**0.7663**

As we know, we wish for the value of AUC and accordingly, pAUC to be as close to 1 as possible (1 marks a perfect classifier), so going by this we choose Convolutional Neural Network Autoencoder as the better model, and model which we will use to train and test data from the Evaluation set. If we consider only this measure, the CNNAE with L2 regularization is the clear winner. We can also see that Batch normalization improves the results of the VAE, but the results are still not as good as those of the CNNAE. Additionally, the metrics for both CNNAE and CNNAE with L2 regularization did not change at all when adding batch normalization, or skipping connections.

When it comes to performance, the runtime cannot be ignored. The VAE takes about 60 minutes to train. As expected, training is faster when batch normalization is added, with it being about 50 minutes. The CNNAE, on the other hand, has a training time of just under 15 minutes (this does not change when we add regularization, which is expected).

Overall, the CNNAE with L2 regularization proved to be the best choice for this task, providing the highest AUC (and pAUC) while requiring the least amount of training time.

6 Conclusion

As mentioned above, we chose the CNNAE with L2 regularization because it's clearly the best performing of all the methods we tested. The final score achieved by this model on Kaggle is 0.77125. This is a very satisfying result, considering that the baseline was only able to achieve a score of 0.54339 and had a runtime that's more than three times as long. All of the written code can be found in a public notebook on Kaggle with all appropriate credits to various sources we used.