AML Lab Report

Challenge 2 - Sound analysis

Name of Student 1: Zachari Thiry
Name of Student 2: Matthieu Ramon
Name of Student 3: Benjamin Salon
Promo: 2023

AML : Advanced Machine Learning
(Spring, 2022)

EURECOM Graduate School and Research Center in Digital Science
Data Science Cursus

17th September 2022

Abstract

Machine learning exercise made during the Algorithmic Machine Learning course at EURECOM.

Based on a collection of sound recordings from different industrial machines, we attempted to classify normal from anomalous sound emissions. We were given only "normal" sound recording and we had to be able to detect anomalous sound recordings when the model is trained. We oriented our research towards a Variational Auto Encoder model using PyTorch and we successfully reached a score of 0.75 Area Under Curve.

Contents

1 Data Analysis

1.1 Introduction to the data set

The given data set is a collection of sound recordings from 5 different industrial sliders. It is extracted from the bigger challenge DCASE 2020 Task 2. Here only the slider machines are kept. We have a development data set and an evaluation data set. The development data set is composed of three machines of index 0, 2 and 4 while the machines 1 and 3 are in the validation data set.

1.2 Analysis and first thoughts

We start by listening to the recordings on kaggle. We can easily by ear distinguish an anomaly from a normal sample. This distinction can also be done by looking at the mel spectrograms of the sounds:

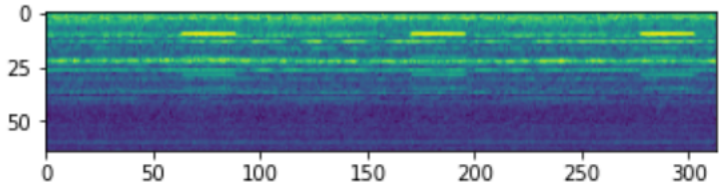

Figure 1: Spectrogram from a normal sample

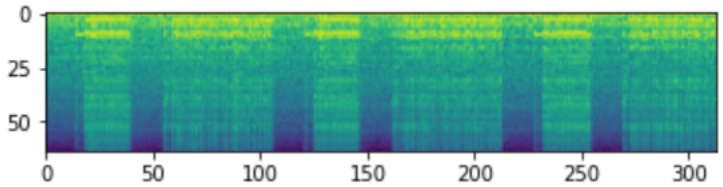

Figure 2: Spectrogram from an anomaly sample

We can see here that the distinction can be also done with the mel spectrograms relatively easily. This leads us towards a image-based machine learning method.

2 Data Pre-Processing

2.1 Data Augmentation

Mel spectrograms are a way to apply image-related machine learning algorithms to a sound data set. To do data augmentation with images we can for example rotate the image, scale it, modify the color or lightning, or add some noise. The nature of the image would not change so the same label could be applied to the original sample and the augmented sample.

Such techniques however don't work with sound analysis : rotating or changing the spectrogram significantly changes the sound it represents.

One traditional way of augmenting spectrographic data set has been introduced by google under the name SpecAugment. It consists in blocking sections of the spectrogram in one or two of the following ways :

- A frequency mask that randomly hides some of the frequencies of the sound sample for the whole duration of the recording. This has the effect of balancing the weight of frequencies and avoid overfitting
- a time mask that masks some portions of the recording.

See below an example of a masked spectrogram :

Figure 3: Spectrogram with both frequency and data masks

2.2 Implementation

To implement our augmentation, we ran into numerous issues which consituted most of our time in this lab. First, we were not familiar with Pytorch, and then our understanding of the common.py functions remain limited today. As such, although we could produce many masks from one sample, we were only able to train our model on one masked spectrogram per initial data point.

3 Model Selection and results

3.1 Choice of a Variational Auto Encoder

We were put on the clue of a VAE from the beginning thanks to the baseline provided by the teachers. With this in mind, we did the necessary research around the VAE and how to use it for anomaly detection we decided to keep the VAE model as it seems to be the state of the art concerning this type of tasks and to tune its parameters, along with the data pre-processing and augmentation.

3.2 Model tuning

As we are, for this challenge, in a deep learning problem, every investigation on new hyper parameters values is very time costly. We decided to give a first approach of hyper parameters tuning with reducing the number of epochs for the training. We tested the different values with a first number of 5 epochs and then we increased at 20 epochs to confirm our value selection. Finally to get our final results and submission we went up to 100 epochs.

We first needed to be sure that it meant something to compare the models with only a low number of training epochs. To do so we registered the performance of the model over the development data set for 5 different initialization of the model. Our results averaged on the different machine were:

No of training	1	2	3	4	5
Average AUC	72.33	72.33	70	72.33	75

Figure 4: AUC on 5 epochs training with different seeds

As the results were substantially constant we started experimenting with the different hyper parameters values. We investigated:

Learning rate	0.1	0.01	0.001
Batch_size	1000	200	100

Figure 5: Values investigated

Which lead us to select our best parameters:

Learning rate	0.001
Batch_size	200

Figure 6: Values kept after investigation

Batch size greatly impact algorithm performance. However, it also significantly increases the run time. The figure below shows the results we obtained running different batch sizes (not displaying results above size 500). Since the results only slightly improve with a batch_size of 100 compared to 200 but the complexity roughly doubles, we decided to stick to a size of 200.

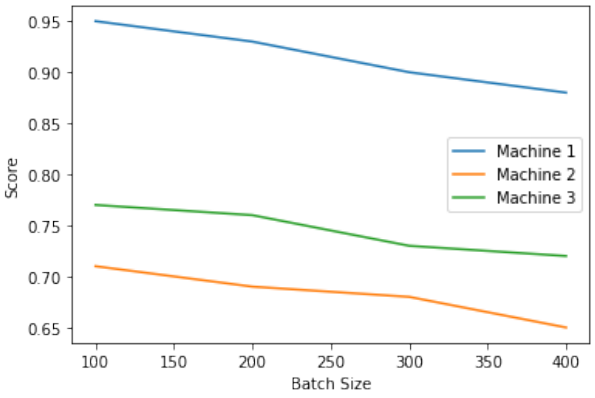

Figure 7: Scores as function of batch size

3.3 Results

After tuning and choosing the best parameters found we got the following scores:

4 Limits reached and self analysis

- **Deep learning challenge** For this challenge, the time was a real issue. Every time we want to validate completely our theories we need at least

Epoch number	5	20	100
Average training AUC	0.82	0.85	100
Test AUC	N/A	N/A	0.752

Figure 8: AUC Scores with retained parameters

one hour of training to get the validation or not of our hypothesis. This is why we decided to go for a parameter selection with low epochs, but we still needed to anticipate the time needed and it is a true key point in the team work organization.

- **PyTorch** It was our first time using PyTorch for all of us in the team. As it is a low level library it took us a lot of time to understand the baseline and code example we could find online.

References

[1] Lecture notes

[2] DemisEOS/SpecAugment