AUDIO ANOMALY DETECTION

Course: Advanced Audio Processing
Contributors: Trung Van, Minh Tran

Table of Contents

1. Introduction

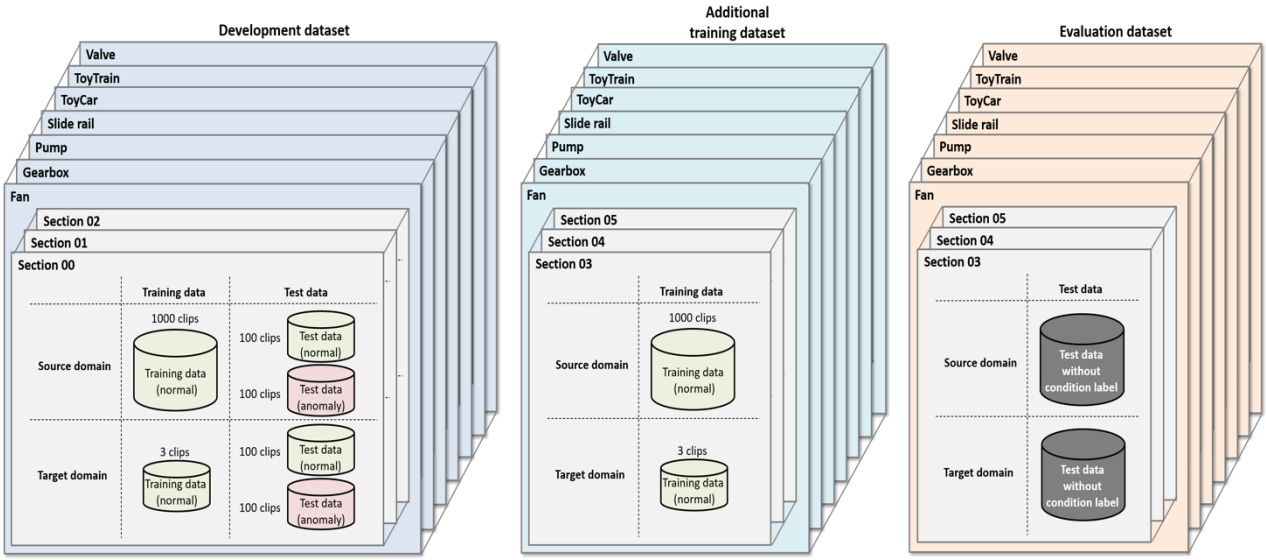
The inception of this project is from DCASE challenge 2021, task 2: Anomalous Sound Detection (ASD) [1]. The idea to identify whether the sound emitted from a machine is normal or anomalous. In the chosen challenge, domain shift was introduced with the goal of improvement in newly observed conditions for the anomaly detection model. As a result, our model will be more robust to real-world application with a large variation in operational conditions.

1. Dataset

The given data is sound clips of seven type of machines with the following specifications:

- Duration: 10 sec
- Single channel, fixed microphone
- Mixed with environmental noise (which was recorded from several real factory environments)

The dataset distribution can be seen from the figure below



The amount of anomalous and from target domain are minor in comparision with data from source domain/ normal machine.

In our implementation, we do not have access to the evaluation dataset so the development was partitioned into train and test dataset for this project.

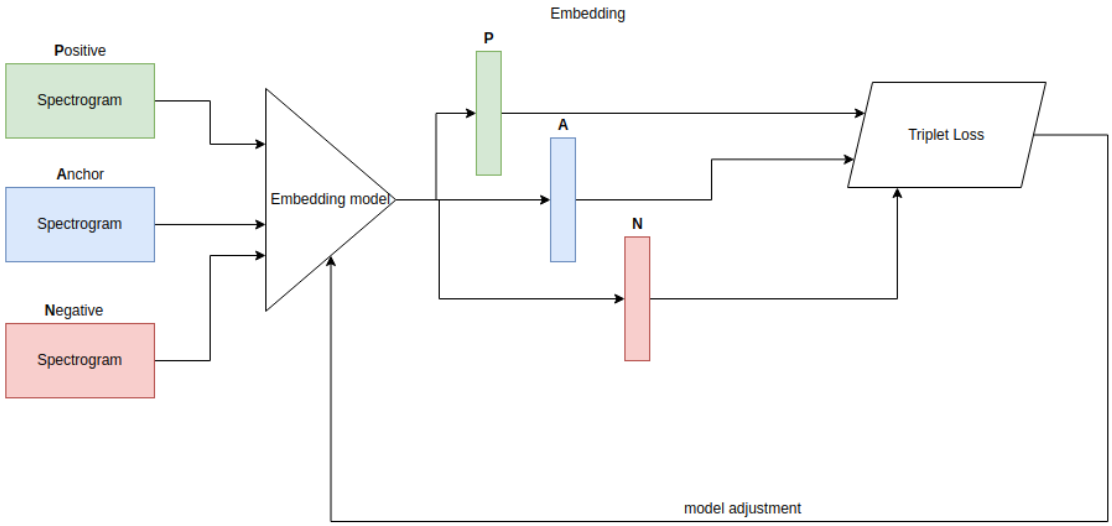
2. Evaluation

The detail information for evaluation can be found from DCASE challenge homepage. In our implementation, we used only AUC to evaluate the model's prediction.

2. Theoretical background

The bird-eye-view of our approach is to get the anomaly detection as an advantageous biproduct from the clustering (based on sections – different operational conditions) our data. The architecture of this project is quite minimal which can be seen from the diagram:

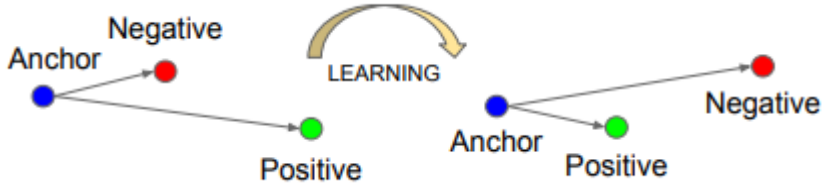
1. Train phase



Train Phase

The embedding model is a CNN-based feature extraction to reduce the initial inputs (spectrogram) down to small latent embedding vectors. The goal of the embeddin model is not only to reduce the dimensions of the data but also to learn the representative information for each extracted sound feature.

During the training phase, Triplet Loss[2] is used to separate the data into different clusters of sections.

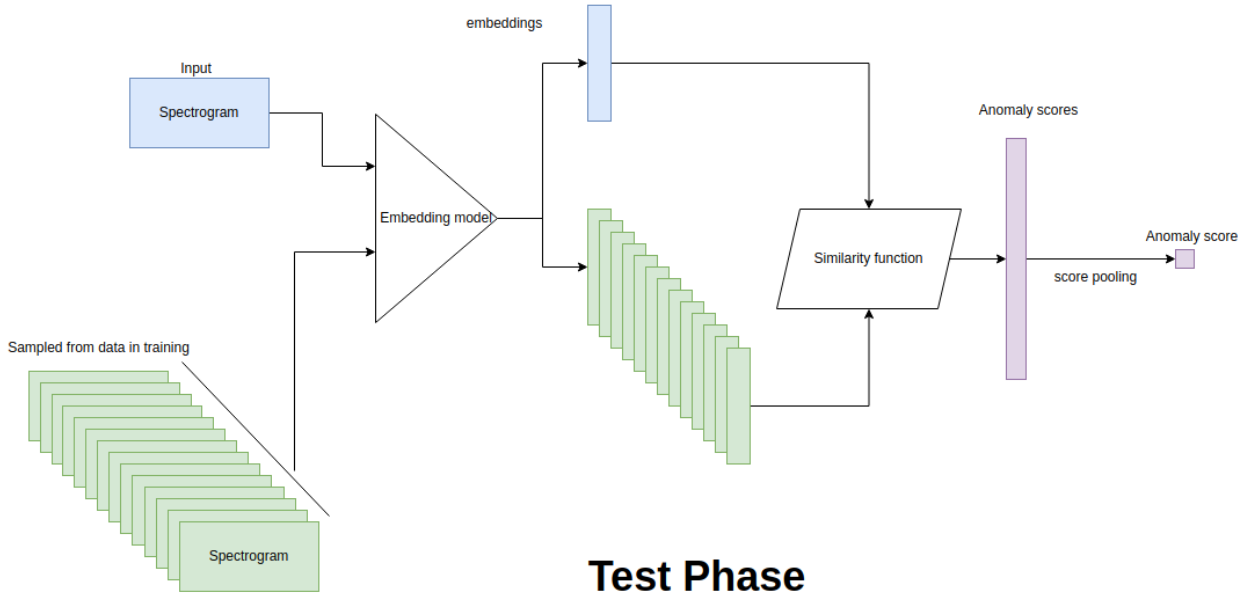


Our model acts like a function that takes two pairs:

- Anchor – Positive -> reduces the distance (or increases the similarity result) between them
- Anchor – Negative -> implements the opposite -> increases the distance (or decreases the similarity result) between them

Iteratate through the dataset with this implementation, we end up with clusters of samples that have the large similarity in the latent space and small similarity with the samples from other clusters. However, the emphasized product of the training phase is our model with the ability to cluster samples and to embed the normal/anomalous information in to the extracted features.

2. Test phase



During the test phase, the embedding model is used to extract features of the test input data and a determined number of same from all sections that were used during the train phase. The extracted features then will be used to calculate the similarity (anomaly score) between the test data and the samled data mentioned above. Pooling a value from the vector of anomaly score gives us the anomaly score of the given input data.

The last step is to set a suitable threshold for anomaly score to optimize the output inference from our anomaly scores in the entire test dataset using AUC.

3. Implementation

The main principle of implementing the method is embedding the signals using convolutional neural networks and calculating the distances from the samples to anomalous/normal sounds to classify them. The classifier is implemented in the following steps:

- The data used in the model is DCASE2021. There are seven different types of machines. Each machine's Dataset has three sections, and each section includes 1003 training clips and 400 test clips (200 anomalous). Thus, in total, we have 3009 training samples and 1200 test samples for each machine.
- To extract the features, calculate the amplitude of the Mel spectrogram of the samples. The other features are extracted from the file names: the section and the label. To avoid repeating the process and increase the model's efficiency, we group the features into rows in DataFrame format. The training DataFrame includes 3009 rows, and the testing DataFrame includes 1200 rows.
- We create the Dataset and Dataloader as follows:
    - The Dataset contains the data from the DataFrame file. The __getitem__() method will return three samples: the current sample, one positive sample in the same section, and one negative sample in a different section for the training dataset. For the testing Dataset, the __getitem()__ method will return the

feature, section, and label of the sample. The Dataset will label each anomalous sample as 1 and each normal sample as 0.

  - The Dataset also has a method sample_to_infer(), which is used to calculate the anomaly score (this will be described later). The method will only work on the training Dataset, and it will return random samples from the Dataset. The number of samples being inferred can be specified in the params.py file.
  - The Dataloader will read Datasets as batches for training efficiency.

- The embedding model consists of:
  - Three convolutional blocks. Each block contains a 2D convolutional layer, a ReLU activation layer, a batch normalization layer, a 2D max-pooling layer, and a 2D dropout layer.
  - A flatten block to convert the features to a 1D vector.
  - A dense layer to downsize the vector.
- To train the embedding model:
  - Load the data from the DataFrame files and create the Datasets and Dataloaders.
  - Instantiate the model.
  - The model will use triplet loss function (nn.TripletMarginLoss()) and Adam optimizer (torch.optim.Adam()).
  - For each epoch, the training Dataset will return the anchor, positive and negative samples (mentioned in the __getitem()__ function of the Dataset). The model will embed these samples to a 1D vector, and these vectors are used in the triplet loss function. The loss function will then carry out backward propagation.
  - After training, the model is then saved for testing purposes.
- We carry out the inference process to test the model and determine the anomaly score. For each sample in the testing set, we use the method sample_to_infer() of the training Dataset to get random samples from it. Then, both the test and inferred samples are embedded using the model. We calculate the distances from the test sample and the inferred samples using nn.PairwiseDistance(), and the mean of all the distances as the anomaly score.
- We use the ROC curve (receiver operating characteristic curve) and AUC (area under ROC curve) to determine the best threshold anomaly score for classification. We will create 100 possible threshold values between the minimum and maximum anomaly scores using np.linspace(). Each threshold value will be used for classification, and the predictions will be given a AUC score using function sklearn.metrics.roc_auc_score(). The greatest AUC score and the corresponding threshold value will be returned.

4. Results

In the file params.py, the hyperparameters of the embedding model is stored. We use the following parameters:

```
params = dict(
    cnn_channels_out_1=256,
    cnn_kernel_1=5,
    cnn_stride_1=2,
    cnn_padding_1=2,
    pooling_kernel_1=3,
    pooling_stride_1=1,

    cnn_channels_out_2=256,
    cnn_kernel_2=5,
    cnn_stride_2=2,
    cnn_padding_2=2,
    pooling_kernel_2=3,
    pooling_stride_2=1,

    cnn_channels_out_3=256,
    cnn_kernel_3=3,
    cnn_stride_3=1,
    cnn_padding_3=1,
    pooling_kernel_3=3,
    pooling_stride_3=2,

    linear_input=399360,
    output_embeddings_length=16,
    dropout=.25
)

sampled_feature_num = 5  # number of feature-embeddings to pick from one section in inference phase
```

Running the entire system on different types of machines with the given parameters, we have the following threshold values and the AUC scores:

- Fan:
  - Threshold value: 1.2533133839347146
  - AUC score: 0.627000000000001
- Gearbox:
  - Threshold value: 0.08053135831494618
  - AUC score: 0.503071931101417
- Pump:
  - Threshold value: 0.8315680045070071
  - AUC score: 0.5
- Side rail:
  - Threshold value: 1.133293143426519
  - AUC score: 0.5000162848767777

The whole system returns the AUC scores in the range 0.5-0.7. Unfortunately, these scores are not very good, as 0.5 means that the model cannot separate classes accurately [3]. Due to time constraints, we also have not been able to train and test the model on all seven machines, making the evaluation of the result less accurate.

There are many ways to optimize the system to increase its performance. We can change the architecture or the hyperparameters of the embedding model. We can also change the

learning rate, the number of epochs, or the optimizer. There are different ways to calculate loss function, infer samples, and calculate loss functions.

Despite the results being not optimal, our system has introduced an approach different from what we learned in the course. We believe that there is much potential to improve the classification ability of the system.

5. Reference

[1] DCASE challenge 2021 – Task 2: Unsupervised Anomalous Sound Detection for Machine Condition Monitoring. https://dcase.community/challenge2021/task-unsupervised-detection-of-anomalous-sounds

[2] Schroff, F.; Kalenichenko, D.; Philbin, J. (June 2015). FaceNet: A unified embedding for face recognition and clustering. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 815–823. arXiv:1503.03832

[3] Narkhede, S. (2021, June 15). *Understanding AUC - roc curve*. Medium. Retrieved March 14, 2022, from https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5