Term Research
Finding Anomalies in Large Unlabeled Acoustic Data

Philip Robinson

Oregon Health Sciences University

December 4, 2019

Abstract

University of Hawaii's Aloha Cabled Observatory (ACO) has approximately 10 years of unlabeled hydrophone data, in need of indexing, labeling, and cleaning; to more easily extract acoustic anomalies. Currently searching the audio recordings is done at human pace by listening to the audio stream, or looking at it's spectrogram, which takes $\sim$ 2-20 minutes for every 5 minutes of audio. This is intractable at scale. Unfortunately, audio data is known to be high dimensional, making it very difficult to label raw input streams. Additionally, the properties of acoustic anomalies are not necessarily known a priori. `NEEDS REFERNCE AND NP-PAPER`

1 Introduction

Scientists, with University of Hawaii's Aloha Cabled Observatory (ACO), have gathered 10+ years of continuous audio for acoustic ocean survey studies. Currently searching the audio recordings is done at human pace by listening to the audio stream, which takes $\sim$ 2-20 minutes for every 5 minutes of audio, depending on experience levels. Even at its fastest, this is intractable at scale. Often ocean hydrophone observatories are used to survey and track cetations vocalizations for population measurement, acoustic analysis for earthquake/tectonic events, and applications in monitoring vessel traffic and activities. The ACO hydrophone data is unlabeled and growing at 1.2 terabytes per year.

> *"Being able to automatically detect whale calls in an un-monitored system can help identify time-series of whale locality to the area. Being able to apply this detection algorithm to a long time-series of sub-sea audio, such as that from the ACO, allows scientists to derive a time-series of whale activity in the area. This can help us study the relationship between migration patterns and known climate events."*
>
> - Kellen Rosburg
> Senior Ocean Computer Specialist
> OOI Cabled Array, APL/UW

In the most general form, it would be extremely useful to have a generic unsupervised acoustic anomaly detector, for hydrophone data. This would allow a human or machine reviewer to focus on only interesting data in the task of event sorting and classification.

2 Model

In prior work on acoustic anomaily detection [1] address the generic task of anomaly detection, over image encoded acoustic anomalies. A Variational Auto-Encoders is expected to behave as an identity function, however reconstruction error can be an indicator of content unseen in the training data. When a Variational Auto-Encoders is trained on typical data, a threshold over an anomaly score can be used for to flag atypical data. The paper proposes a minimization of false positive rate, given a constrained true positive rate, by expressing a threshold of reconstruction error.

Proposed is an anomaly generating network. This is done by fitting the Variational Auto-Encoders 's gaussian manifold on general data, generating an under-specified probability density function (by minimizing KL divergence), then tighter fitting a more descriptive probability density function over the latent vectors of typical data projected into the gaussian manifold. Once both distributions are fit, rejection sampling in the form of sampling from the general distribution values of low probability from the descriptive distribution.

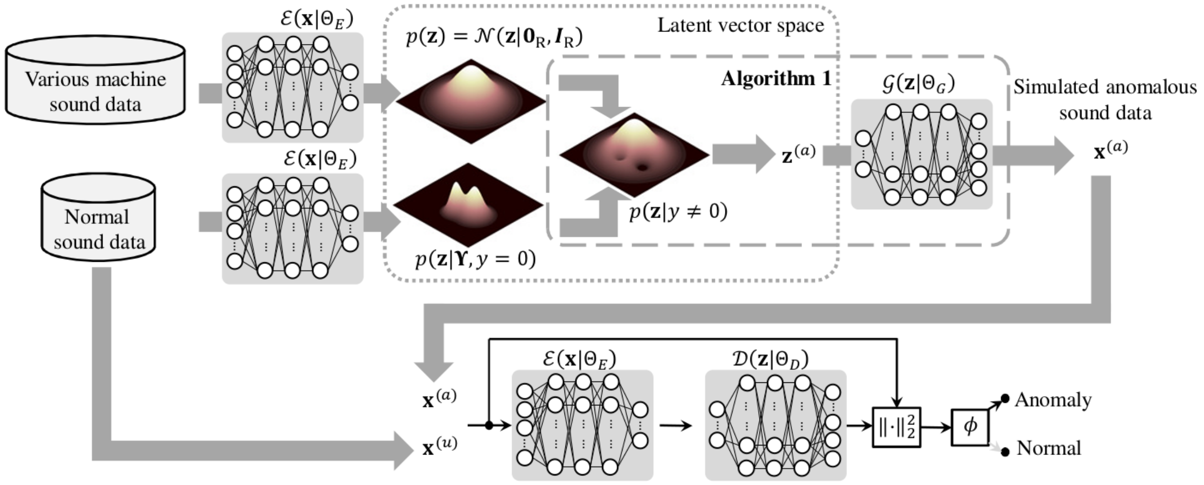
Training is broken up into thee major phases. These phases contribute to fit the generator and discriminator networks, as well as a typical data model (in this case our GMM) for informing rejection sampling.

The first phase updates the parameters of the encoder and the generator for all input data. The goal of this process is to define the general probability density function of the latent space of all data and a mapping to a unit gaussian manifold.

The second phase updates the parameters of the `encoder` and `descriminor` with a loss function that is defined by reconstruction error against a threshold, and an approximation of true/false positive rates (which cannot be calculated directly). This phase requires sampling acoustic anomalies from the unit gaussian and rejecting likely events measured by the specific probability density function, through the generator network. For the ACO typical data can be sampled from the evenings or low traffic months, as cetations tend to not be as vocal in the evenings; this strategy is also resilient to accumulative damage/drift of the equipment over time.

The third phase is to update the specific probability density function modeled by a `GMM`. This is repeated as necessary.

Discrimination is accomplished by learning an appropriate threshold for the likelihood of a latent representation of an event.


Various machine sound data
Normal sound data
$\mathcal{E}(\mathbf{x}|\Theta_E)$
$\mathcal{E}(\mathbf{x}|\Theta_E)$
$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}_R, \boldsymbol{I}_R)$
Latent vector space
Algorithm 1
$p(\mathbf{z}|y \neq 0)$
$p(\mathbf{z}|\mathbf{Y}, y = 0)$
$\mathbf{z}^{(a)}$
$\mathcal{G}(\mathbf{z}|\Theta_G)$
Simulated anomalous sound data
$\mathbf{x}^{(a)}$
$\mathbf{x}^{(a)}$
$\mathbf{x}^{(u)}$
$\mathcal{E}(\mathbf{x}|\Theta_E)$
$\mathcal{D}(\mathbf{z}|\Theta_D)$
$\|\cdot\|_2^2$
ϕ
Anomaly
Normal

3 Topics

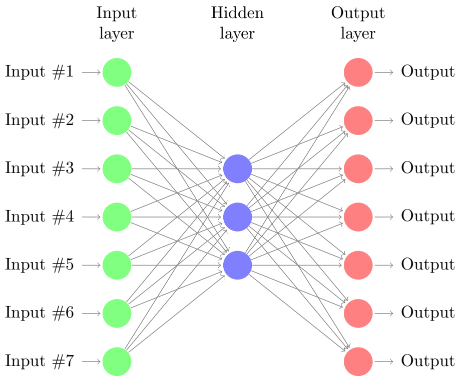
Due to the complex nature of this architecture, it is important to provide an understanding and background of supporting topics. This section introduces the pre-requisite knowledge to understand the final model.

3.1 Auto-Encoders

Auto-Encoders are an unsupervised neural network model that attempts to constrain an identity function. This is usually done by optimizing the weights of the neural network to reconstruction error, while passing through a low dimensional `bottle`.

$$J^R = ||Input - Output||_2^2$$

The `bottle`, represented as a narrowing of nodes in the neural network, expresses a compressed representation of the original content wrt it's neighboring encoding and decoding networks. These networks can be thought of as learning a lossy compression and decompression functions. The lossy characteristic of these mapping functions is expressed in image data as a blurry recreation of the original content.



../ae.py

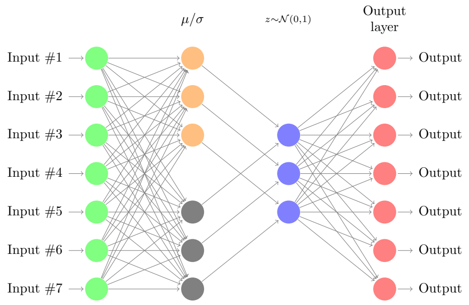
```
class AutoEncoder(BottleNetwork):
    _reconstruction_error = partial(F.mse_loss, size_average=False)
    # _reconstruction_error = partial(
    #     F.binary_cross_entropy, size_average=False
    # )

    def __init__(
        self, *, bottle_size, data_shape, EncoderType, DecoderType,
        encoder=None
    ):
        super().__init__(bottle_size=bottle_size, data_shape=data_shape)
        self._encoder = encoder if encoder \
            else EncoderType(bottle_size=bottle_size, data_shape=data_shape)

        self._decoder = DecoderType(
            bottle_size=bottle_size, data_shape=data_shape
        )

    def parameters(self):
        return chain(self._encoder.parameters(), self._decoder.parameters())

    def encode(self, X):
        return self._encoder(X)

    def decode(self, z):
        return self._decoder(z)

    def forward(self, X):
        h = self.encode(X)
        Y = self.decode(h)
        return Y

    @classmethod
    def reconstruction_error(cls, Y, X):
        return cls._reconstruction_error(Y, X)

    @classmethod
    def loss(cls, X, *args):
        Y, *_ = args
        return cls.reconstruction_error(Y, X)
```

3.2 Variational Auto-Encoders

Variational Auto-Encoders are similar to Auto-Encoders, however they restrict the form of manifold forming the `bottle` to a gaussian manifold. It is important to note that trained Variational Auto-Encoders can be split at the `bottle`, and used like a generative model. This is done by sampling from a unit gaussian; the samples is interpreted as a latent representation of an input from the `encoder`. The `decoder` is then used on this sample to generate new content that would have been encoded as the latent representation.

In order to train this style of network, the loss function is complimented by Kullback Leibner divergence, to constrain the latent space to a gaussian manifold.

$$J^{KLR} = J^R + KL(z||\mathcal{N}(0,1))$$



../vae.py

```
class VariationalAutoEncoder(AutoEncoder):
    def __init__(
        self, *, h_size, z_size, data_shape,
        EncoderType, DecoderType, encoder=None
    ):
        super().__init__(
            bottle_size=h_size, data_shape=data_shape,
            EncoderType=EncoderType, DecoderType=DecoderType,
            encoder=encoder)
        self.z_size = z_size
        self.h_size = h_size

        self.mu = nn.Linear(h_size, z_size)
        self.logsigma = nn.Linear(h_size, z_size)
        self.eta = nn.Linear(z_size, h_size)

    def parametrs(self):
        return chain(
            super().parameters(),
            self.logsigma.parameters(),
            self.eta.parameters(),
            self.mu.paramerters()
        )

    @classmethod
    def _reparameterize(cls, mu, logsigma):
        std = logsigma.mul(0.5).exp_()
        # return torch.normal(mu, std)
        epsilon = torch.randn_like(mu)
        z = mu + std * epsilon
        return z

    def bottle(self, h):
        mu = self.mu(h)
        logsigma = self.logsigma(h)
        z = self._reparameterize(mu, logsigma)
        return z, mu, logsigma

    def decode(self, z):
        h = self.eta(z)
        return self._decoder(h)

    def _sample(self, n):
        z = torch.stack([GaussianSample(self.z_size)
                         for _ in range(n)])
        return z

    def generate(self, n=1):
        z = self._sample(n)
        with torch.no_grad():
            h = self.eta(z)
            return self._decoder(h)

    def forward(self, X):
        h = self.encode(X)
        z, mu, logsigma = self.bottle(h)
        Y = self.decode(z)
        return Y, mu, logsigma

    @classmethod
    def _KL_loss(cls, mu, logsigma):
        return -0.5 * torch.mean(
            1 + logsigma - mu.pow(2) - logsigma.exp())

    @classmethod
    def _vae_loss(cls, X, *args):
        Y, mu, logsigma, *_ = args
        return cls._KL_loss(mu, logsigma) + \
```

3.3 Rejection Sampling

Rejection sampling is a technique used in many statistical models. Many distributions are difficult to sample from, however simple to evaluate likelihood against. A second distribution whose domain is unbiased wrt the original distribution may be elected to act as a proxy for sampling.

The simplest example of this is in attempting to sample points from a unit circle. There exist known algorithms to sample from a uniform distribution. A two dimensional uniform distribution is equivalent to sampling from a rectangle. Rejection sampling can be used to sample from the unit circle by electing a 2 dimensional uniform distribution that encompasses the area of the target circle, and rejects points under the constraint that $x^2 + y^2 \leq 1$.

This relates to Variational Auto-Encoders because there exists known algorithms for sampling from gaussian distributions. We can elect any statistical model to describe typical data, that shares an unbiased domain with a gaussian. This typical distribution can then be used to form a likelihood threshold for rejection criteria. This strategy informs the core of the proposed model, further explained by the Neyman-Pearson Lemma.

3.4 Neyman-Pearson Lemma

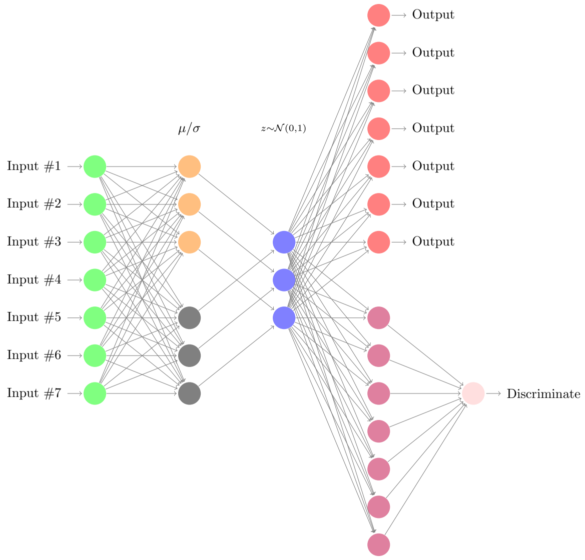
The Neyman-Pearson Lemma states that the likelihood ratio is the 'uniformly most powerful discriminator' for a statistical hypothesis test. This is leveraged into forming a loss function by increasing the reconstruction error of anomalous data, and decreasing the reconstruction error of typical events in data.

$$J^{NP} = FP - TP$$

Theoretically a discriminator network leveraging this loss function, with known anomalous and typical data, will become better identity function for typical data and increasingly poor at reconstructing atypical data.

4 Gumiho Networks

Although this term isn't in the common nomenclature, a gumiho network refers to a network that has more than one tail, generating output. The intent is to use different loss functions depending on which network output is triggered. For Auto-Encoders and Variational Auto-Encoders this allows for the learned encoding function to generate the same manifold, while satisfying very different loss enforced constraints. It is helpful, although not necessary, to choose non-competing loss functions.



Since the discriminator network uses a threshold over reconstruction error, the Variational Auto-Encoders (that uses only recognition error) is not considered a non-competing loss function. This conditional training, dependent on the evaluation path lends itself to `pytorch` [2] over `keras`. This will be discussed in greater detail in the next section.

5 Implementation

Systems like `keras` and `tensorflow` use a precompiled execution graph, to facilitate more predictable back propogation. For this model, there is a parameterized discriminating function in the `bottle` of the network. Additional complexity is added, due to the sharing `encoder` variables across multiple `decoder` networks. `pytorch` [2] doesn't restrict the model to precompiled execution graphs and allows for use of the `torch.no_grad()` context manager in order to restrict when variables can be treated to not need back propogation.

The user provides an `encoder`, `decoder`, the interfacing dimension size h, the `bottle` dimmension size z, and the count of mixtures for the `GMM` M. The `encoder` and `decoder` are expected to reduce the input to and from a linear layer of size h. A linear `bottle` layer is provided to map from h to z dimmensions and extract μ and σ to inform KL loss by reparameterization. The `decoder` is then preceded by a provided layer η that maps back from z to h dimmensions. This abstraction layer makes it much easier to modularize the network.

The code represents a direct inheritance principled on object oriented design of machine learning models. Descriminator Network is a Conditional Generating Network, is a Gumiho Network, is a Variational Auto-Encoders , is a Auto-Encoders.

5.1 Gumiho Network

The Gumiho Network is the first unfamilure network abstraction, and therefore starts the descriptive sections of this text.

The Gumiho Network is a Variational Auto-Encoders with multiple goal states. A user can add additional decoders and cooresponding loss functions. This allows a trained model to develop an embedding optimized, not only for reconstruction but, for various additional goals like classification. The add_tail(self, network, loss) method requires the network can be fed by an h dimensional linear layer.

../gumiho.py

```
class GumihoNetwork(VariationalAutoEncoder):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.tails = {}
        self.losses = {}
        self.add_tail(None, self._decoder, self._vae_loss)

    def parameters(self):
        tails_params = (
            self.tails[key].parameters()
            for key in self.tails
            if key is not None  # because it is alread added
        )
        return chain(super().parameters(), *tails_params)

    def forward(self, X, *, tail):
        h = self.encode(X)
        z, mu, logsigma = self.bottle(h)
        Y = self.decode(z, tail=tail)
        return Y, mu, logsigma

    def decode(self, z, *, tail=None):
        return self.tails[tail](z)

    def _generate_from_z(self, z, tail=None):
        with torch.no_grad():
            return self.decode(z, tail=tail)

    def generate(self, n, *, tail=None):
        z = self._sample(n)
        return self._generate_from_z(z, tail)

    def add_tail(self, key, network, loss):
        self.tails[key] = nn.Sequential(self.eta, network)
        self.losses[key] = loss
```

5.2 Conditional Generating Network

In a Variational Auto-Encoders , the embedding is mapped to a gaussian manifold. This means that sampling from a unit gaussian and passing data through a decoder will generate results coorespnding to the decoder's goal. When used with the tail associated with the reconstruction task, this produces viable input data. the Conditional Generating Network allows condition to be provided to the sampled value and the tail elected to be specified by the user. For the final model this is a mechanism to allow the GMM to be used for conditioning and the reconstructor to be used to generate anaomolies as found in the reference materials.

../gumiho_descriminator.py

```
class CondGeneratorNetwork(GumihoNetwork):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.conds = {}
        self.add_cond(None, self._none_cond)

    @classmethod
    def _none_cond(cls, x):
        return True

    def _sample(self, n, *, tail=None):
        def _local_gen():
            while True:
                z = GaussianSample(self.z_size)
                if self.conds[tail](z):
                    yield z

        g = islice(_local_gen(), n)
        f = [identity.remote(_) for _ in g]
        z = ray.get(f)
        Z = torch.stack(z)
        return Z

    def add_cond(self, key, cond):
        self.conds[key] = cond

    def generate(self, n, *, cond=None, tail=None):
        z = self._sample(n, tail=cond)
        return self._generate_from_z(z, tail=tail)
```

5.3 Descriminator Network

The Descriminator Network models the machine specified [1] for unsupervised anomaly detection. There are three tails, one for training the GMM, one for discrimination, and one for reconstruction. The code for this model is found in the repository instead of this paper due to its complexity.

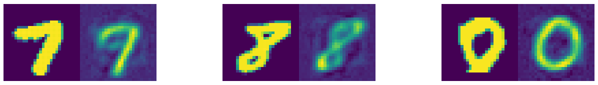
6 Data

The model is build to be agnostic to types of encoder and decoder. The expectation of the user is to provide models appropriate for their data. It is also expected that a loading coroutine is provided. The coroutine needs to take in the count of a batch in order to be trained in a consistant way.

6.1 MNIST

The MNIST data hopes to example hand written single digit entries. The original intent is to inform algorithms of had written digit recognition for the postal service. For this term project, MNIST was used to more consistently and reproducibly explore model building.

In this trained model, the digits listed as $\{1, 7, 4\}$ were labeled as anomalous. Any other digit were described as typical events. The expectation of a trained network is to generate high accuracy reconstructions of $\{2,3,5,6,8,9\}$ and low accuracy reconstructions of $\{1,7,4\}$ compile.



The images above are sources and their paired results from passing the MNIST data through the trained Variational Auto-Encoders with loss of mean squared error.

../ae.py

```
def data_initializer(*,
    censored=[2, 3, 4, 5],
    atypical=[1, 7],
    various=0,
    data_shape
):
    HERE = Path(".")
    _ = torch.utils.data.DataLoader(
        datasets.MNIST(
            HERE,
            train=True,
            download=True,
            transform=transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize((0.0,), (1.0,))
            ])),
        shuffle=True
    )

    typical_stream = cycle(
        x.squeeze()
        for x, y in _
        if not (y in censored or y in atypical)
    )

    def to_batch(iterable):
        while True:
            n = (yield)
            batch = torch.stack(
                [x.view(data_shape) for x in islice(iterable, n)]
            )
            yield batch

    if not various:
        return to_batch(typical_stream)
    else:
        various_stream = cycle(
            x.squeeze()
            for x, y in _
            if (y in censored)
            or (y in atypical and not randint(0, various))
            or (y not in censored and y not in atypical)
        )
        return to_batch(typical_stream), to_batch(various_stream)
```

6.2 Aloha Cabled Observatory

The ACO data is gathered at N22°45.110′ W158°00′. The recordings are encoded as, custom, variable bit-width raw peak sensor readings. Each file is saved as a 5 minute duration (barring hardware related problems), named with it's datetime stamp. The set elected for this project was 44100 samples per second. The developed ACOio.py library allows simple manual indexing and datetime-search of the data.

```
from aco import ACOio, datetime, timedelta

loader = ACOio('./basedir/')
target = datetime(
    day=18, month=2, year=2016,
    hour=7, minute=55
)

src = loader.load(datetime)
snip = src[
  timedelta(seconds=7):
  timedelta(seconds=11)
]
snip.View()
```

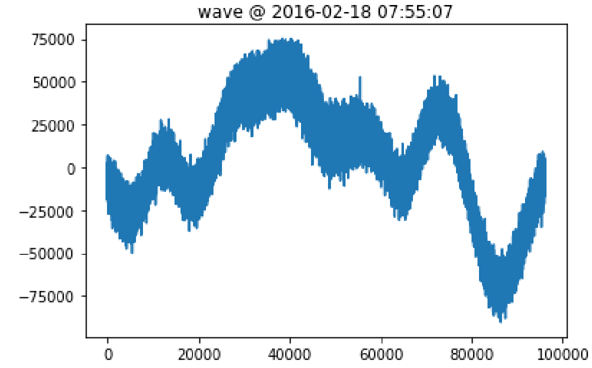


Figure 1: Raw Data

It is visible, from (Figure 1) that the direct current gain is not centered at zero, nor trivially accumulative. This is a consequence of changes in atmospheric pressure, due to the ocean's motion, effecting the signal.

It is also not obvious this track has a vocalization, highlighted in (Figure ??). This pattern is indicative of high amounts of noise, and is expected for all samples.

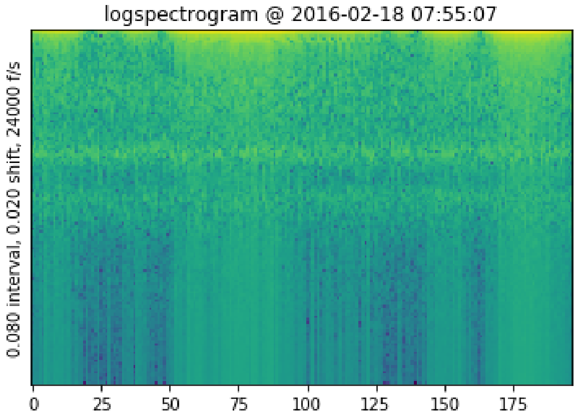


Figure 2: Raw Spectrogram

Inspired by these plots, and studies in signal processing, the audio track can be represented as an image. Expressing acoustic signals as spectrogram and mel-frequency cepstrum algorithms, is a common way to enable these models. This representation lends itself to many deep learning models. Additionally, this allows reasonable evaluation of a target model by using any well studied image dataset. For this project the target dataset is described in the next subsection.

7 Results

Unfortunately, this model never completely worked. I believe that the implementation is very close, however there are training steps that are not updating as expected. I believe that this is consequent of a lack of knowledge about `pytorch` [2] over the actual model. The code is written in a very clean and explainable way that mirrors the writings of the research paper. This will make the project much easier to source instruction and advise to improve. Additionally, in pursuit of a working solution the model is implemented to process over streaming data, and leverages concurrency libraries for the most time expensive sections. The training time of the model is currently on the order of 6 hours for a modern laptop without CUDA support.

The code in it's current state can be found on cslu's gitlab repository[1]

References

[1] Yuma Koizumi, Shoichiro Saito, Hisashi Uematsum Yuta Kawachi, and Noboru Harada. Unsupervised detection of anomalous sound based on deep learning and the neyman-pearson lemma. 2018.

[2] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NeurIPS Autodiff Workshop*, 2017.

[1] `repo.cslu.ohsu.edu/probinso/hydra-anomaly-autoencoder.git`