Tampere University

Manu Harju

TWO STAGE SYSTEM FOR ANOMALOUS SOUND DETECTION IN INDUSTRIAL ENVIRONMENTS

Using few-shot learning in anomaly detection

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiners: Annamaria Mesaros
Tuomas Virtanen
April 2022

ABSTRACT

Manu Harju: Two stage system for anomalous sound detection in industrial environments
Master of Science Thesis
Tampere University
Signal processing and machine learning
April 2022

Machine breakdowns and maintenance breaks cause costly downtime in factories and power plants. Recognizing a breaking machine before the actual breakdown can reduce the downtime and size of the damage. The existing condition monitoring systems are usually based on measuring the vibrations in the machines. In industrial environments the acoustic properties are relatively homogeneous during normal operation, and machine failures cause change in those properties. Therefore a change in acoustic conditions reflects an anomalous event that can be detected through analysis of audio signals at the scene. However, sometimes normal operation like talking or door slamming can cause a significant change in the acoustic conditions, and those should be ignored.

This thesis presents a two-stage acoustic anomaly detection system. The motivation behind using two stages is to offer the operator a possibility to silence certain anomaly types. This makes it possible to ignore normal events that are anomalies from acoustic point of view but do not indicate a need for alarm.

Keywords: anomaly detection, environmental audio, few-shot learning, neural networks

TIIVISTELMÄ

Manu Harju: Kaksivaiheinen poikkeavien äänien havaitsemisjärjestelmä teolliseen ympäristöön
Diplomityö
Tampereen yliopisto
Signaalinkäsittely ja koneoppiminen
Huhtikuu 2022

Laitteiston rikkoutumisista johtuvat katkot voimaloiden ja teollisten laitosten toiminnassa ovat usein kustannuksiltaan kalliita. Laitevioista johtuvien seisokkien pituuksia ja aiheutuvia kustannuksia on mahdollista pienentää tunnistamalla laitteiston huollontarve ennen rikkoutumista. Saatavilla onkin jo valmiita järjestelmiä koneiden kunnon tarkkailuun, ja suurin osa saatavillaolevista järjestelmistä perustuu erilaisten värinöiden mittaamiseen. Teollisessa laitoksessa äänimaiseman ominaisuudet säilyvät yleensä suhteellisen vakiona normaalin toiminnan aikana ja muutos koneen toiminnassa usein ilmenee myös akustisesti, jolloin laiterikko tunnetussa ympäristössä on mahdollista havaita myös äänisignaalien analyysin avulla. Toisaalta joskus myös tavallinen toiminta aiheuttaa merkittäviä muutoksia äänimaisemassa, ja järjestelmän tulisi osata tunnistaa ja jättää huomiotta tapaukset kuten esimerkiksi ihmisten puheet ja ovien kolahdukset.

Tässä diplomityössä on esitetty kaksivaiheinen järjestelmä äänisignaalien poikkeavuuden havaitsemiseen. Esitetty kaksivaiheinen menetelmä tarjoaa järjestelmän käyttäjälle mahdollisuuden hiljentää tietynlaisista äänipoikkeamista aiheutuvat hälytykset, jolloin normaalista toiminnasta aiheutuvat poikkeavat äänet voidaan jättää huomiotta.

Avainsanat: poikkeamien havaitseminen, äänianalyysi, neuroverkot

PREFACE

This work was completed while working in the Audio research group of Tampere University. I would like to thank my supervisor Annamaria Mesaros for guidance and patience throughout the whole process. Furthermore, I want to thank Ville Laukkanen from Ind-Meas for showing the insights of the power plant industry. Finally, I would like to thank my friends and family for their support.

Tampere, 27th April 2022

Manu Harju

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

AE	Autoencoder
AF	Availability Factor
AUC	Area under curve
CNN	Convolutional neural network
FFT	Fast Fourier Transform
FOR	Forced outage rate
kfE	Energy failure factor
MAE	Mean absolute error
MSE	Mean square error
ReLU	Rectified linear unit
SGD	Stochastic gradient descent
VoLL	Value of lost load

1 INTRODUCTION

Deciding about the maintenance schedule of industrial machines is a problem that every operating factory and power plant has to solve. Servicing equipment too often causes unnecessary breaks in production, whereas without proper maintenance machine breakdowns can occur. Machine downtime is costly, and should be minimized.

One way to approach the problem is to monitor the condition of the machines, and try to schedule preemptive maintenance according to the measurements. A commonly used method to examine the condition of a machine is to measure vibrations in critical parts. However, it is also possible to use acoustic monitoring to gain insight into the machine condition.

The work in this thesis concentrates on detecting anomalous events based on captured sound. Industrial sites are loud environments and the acoustic properties of the environment will stay within a known range during normal operation. An anomalous event often causes a change in the acoustic conditions, which could be then detected by audio processing methods. However, some abnormal sounds may be caused by normal operation, and the system should be able to distinguish true anomalies from those.

This thesis proposes a two-stage system consisting of an anomaly detector and a few-shot classifier for a practical approach to condition monitoring using acoustic monitoring. The purpose of the first stage is to detect the abnormal conditions. When such a case occurs, the second stage is used to classify whether the anomaly should cause an alarm or not. Using few-shot learning in the second stage allows the operator to teach the system about the false alarms.

Chapter 2 presents the motivation and theoretical background for the work. Used data and the implementation are discussed in more detail in Chapter 3. Chapter 4 presents the results and discussion, and the conclusions are collected in Chapter 5.

2 BACKGROUND

2.1 Motivation

The main motivation behind this study lies in preemptive failure detection in real industrial systems, especially in thermal power stations. Unexpected failures in such systems cause expensive downtime, that could have been usually avoided with proper maintenance. However, maintenance breaks also stop the production, and having maintenance breaks too often causes unnecessary costs. Occasionally a change in the machine condition causes an audible difference in the sounds it makes. This work attempts to harness audio analysis to recognize unexpected audio signals in industrial environments.

Costs caused by breakdowns and the reliability of power plants are measured continuously to understand the effects of failures and interruptions, and how the performance of the plants can be maintained and improved. The value of lost load (VoLL) is the price estimate of what the customers would be willing to pay to avoid disruptions in their electricity services [1]. The cost varies between countries, and in Finland it is estimated to be 28 000 €/MWh [2].

Analysing power plant performance and reliability can be done using various metrics, and availability factor (AF) is one of the standard measures [3]. The availability factor is defined as

$$AF = \frac{T_A}{T_P}, \tag{2.1}$$

where T_A and T_P denote available and period time, respectively. Power plants that require less maintenance have higher AF, which makes it a useful measure for comparing different power generating systems [4].

For a thermal power plant another measure can be derived from the amounts of energy. The energy failure factor is defined as

$$kfE = \frac{E_f}{E_f + E_d}, \tag{2.2}$$

where E_f and E_d are the loss in energy production and the actual produced energy, respectively [5].

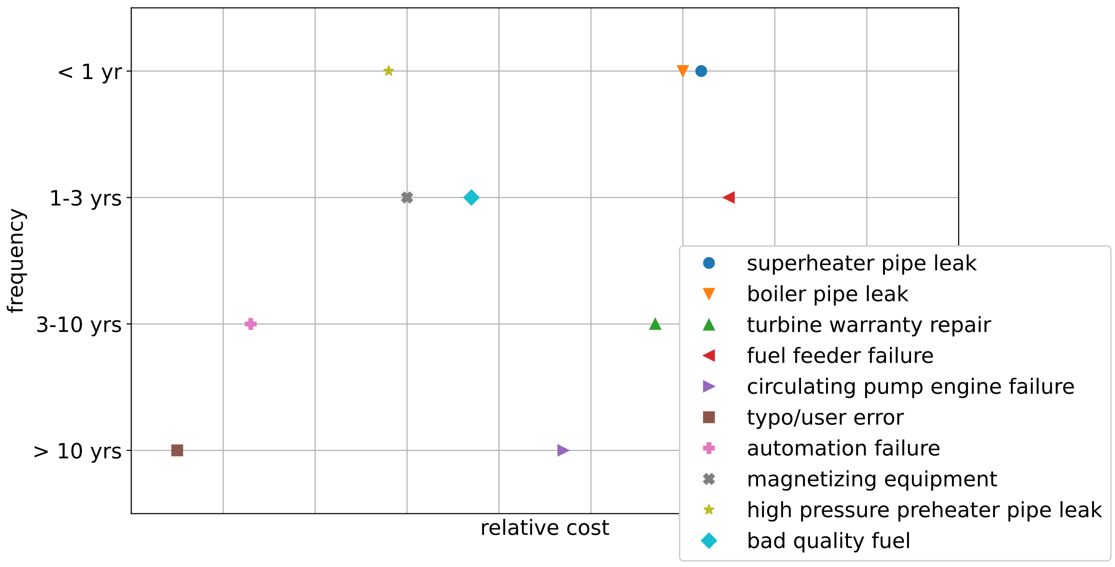

Figure 2.1. *Occurrence rates against costs plotted for different failure classes. Adapted from [6].*

Forced outage is the shutdown condition of a power station, transmission line or distribution line when the generating unit is unavailable to produce power due to unexpected breakdown. Forced outage can be caused by for example equipment failures, disruption in the power plant fuel supply chain or an operator error to mention a few. Forced outage rate (FOR) of a power station unit is the probability that the unit will not be available for service when required. FOR can be calculated as the fraction of the number of hours the unit is on forced outage and the total number of hours in a year [3]. The forced outage rate depends on power plant and fuel type. For coal power and nuclear power the given rates are 4.2 % and 2.1 %, respectively [2].

In thermal power plants decreases in availability are usually caused by failures in the boiler. The most common types of failures are leaks in evaporator and superheater. Moreover, the costs of these failures are relatively high [5, 7]. These findings suggest prioritizing on boiler related problems in practical implementations. Furthermore, Figure 2.1 shows that the problems related to superheater and boiler pipes are also expensive.

Saarinen [8] has collected statistics of failures and interruptions for a sawmill power station. The plant is burning some parts of the wood as fuel, and oil as reserve. Table 2.1 lists a summary of the data. The data consists of events during one year, and the numbers are relatively high as every interruption is counted. There are four boilers in the sawmill power plants: 25 MW steam boiler, 14 MW hot water boiler, and two spare boilers. Half of the fuel problems are related to the steam boiler of the power plant. According to Saarinen [8], the interviews suggest that the most critical parts are fuel processing and ash transfer.

Table 2.1. *Failure statistics for Vilppula sawmill power station in 2016.*

Failure class	count	pct
Fuel (combined)	70	44.30 %
Boiler (combined)	33	20.89 %
Ash (combined)	32	20.25 %
Water (combined)	9	5.70 %
Turbine	9	5.70 %
Generator	0	0.00 %
Reserve power generator	1	0.63 %
District heating	3	1.90 %
Reduction	1	0.63 %

Table 2.2. *Helen Salmisaari power plant failure cost distributions.*

Failure class	Plant A	Plant B
Fuel handling	47 %	29 %
Mechanical maintenance	17 %	2 %
Electric and automation maintenance	30 %	7 %
Material checks	6 %	14 %
Facility development	0 %	31 %
Power plant chemistry	0 %	16 %
Process chemistry	0 %	0 %
Facility operation	0 %	1 %
Condition monitoring	0 %	0 %
Lubrication maintenance	0 %	0 %
Outside factor	0 %	0 %

Table 2.2 shows the distributions of failure costs for two power plants operated by Helen in Salmisaari. Total costs for plants A and B were 710 091 € and 10 174 107 €, respectively [5]. The numbers suggest that in Salmisaari the fuel handling and processing is critical with respect to the total cost. However, this is in line with the other findings [5].

Many of these failure types are due to mechanical breakdowns in the plant equipment, and it is likely that the sounds produced by breaking machinery differ from the sounds during normal operation. For example, failure in a turbine, cavitation in a pump, or excess mechanical wear in machines can cause noticeable change in acoustic conditions. Furthermore, this gives motivation towards developing a system for acoustic monitoring.

Finally, normal operation may also cause abnormal sounds, for example door slamming, tool dropping and people talking to mention a few. These events should be ignored. The objective of this work is to develop an anomaly detection system that can recognize unusual acoustic conditions, and is able to categorize the detected unusual ones. The abnormal sounds that are not of concern should be detected by the system as anomalies, but recognized as non-alarm triggering events.

2.2 Related work

Monitoring systems for industrial environments already exist to some extent as commercial products. For example, AuresSound[1] by APL Systems offers audio based condition monitoring, aiming for customers in energy and process industries. In addition, a Finnish company NL Acoustics[2] already has a product for industrial sound monitoring. Rosemount Inc has patented methods related to acoustic monitoring with wireless devices in industrial environments [9]. Algorithmica Technologies[3] provides a neural network based solution to monitor data from industrial environments, but the solution contains nothing related to audio.

2.3 Neural networks

2.3.1 General concepts

Neural networks are widely used models in modern machine learning. The target is to approximate some mapping f between its inputs and outputs. For example, a classifier is the function $y = f(\mathbf{x})$ that maps the inputs $\mathbf{x}$ to the class y. [10]

A feedforward network is a neural network without feedback connections. Usually a feedforward network is a composition of several functions, and the functions are called the layers of the network. For example, if a feedforward network f^* consists of three chained layers, we can write $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. This kind of chaining arrangement is the most commonly used structure in feedforward networks. The length of the chain is called the depth of the model, and the term deep learning relates to this terminology. A layer whose outputs are not outputs of the model is called a hidden layer. A layer is usually a vector valued function, but instead of a single multivariate mapping a layer can be thought as many similar units representing a vector-to-scalar functions acting in parallel. These units are thought to resemble real neurons in the sense that they get their inputs from other units and calculate their own activations, hence the name neural network. In addition, some choices for the functions $f^{(i)}$ are motivated by neuroscience. However, neural networks should be thought of a tool for function approximation instead of a model for brain. [10]

[1]https://www.apl.fi/

[2]https://nlacoustics.com/

[3]http://www.algorithmica-technologies.com/

2.3.2 Activations and output types

The output of a single unit is often called activations, and they are computed by applying some activation function to the weighted sum of the inputs and the bias term. The output of the jth fully connected layer can be written as

$$\mathbf{y}_j = g_j(\mathbf{W}_j\mathbf{x} + \mathbf{b}_j), \tag{2.3}$$

where $\mathbf{W}_j$ is the weight matrix, $\mathbf{b}_j$ is the bias vector, and the activation function g_j is usually acting element-wise on its input vector. [10]

Since the composition of affine transforms stays affine, it is useful in many cases to choose a nonlinear mapping for the activation. The usual recommendation in modern neural network architectures is the rectified linear unit (ReLU), defined by

$$g(z_i) = \max(z_i, 0), \tag{2.4}$$

where z_i is the input of the ith activation. [10].

In some cases it may be reasonable to use different activation functions. In particular, the activations of the last layer represent the output of the network [10]. In the case of binary classification a common choice for output is the logistic sigmoid function

$$\sigma(z_i) = \frac{1}{1 + \exp(-z_i)}. \tag{2.5}$$

The output of the sigmoid function is between 0 and 1, and the network output can be interpreted as the probability of the input belonging to the target class. For multi-class single-label classification the output of the network should represent a probability distribution. This can be done with softmax activation, defined by

$$\operatorname{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_k \exp(z_k)}. \tag{2.6}$$

In other words, the inputs of the softmax are exponentiated and then normalized [11]. This guarantees that the outputs are always positive and their sum is 1. The hyperbolic tangent is defined by

$$\tanh(z_i) = \frac{\exp(z_i) - \exp(-z_i)}{\exp(z_i) + \exp(-z_i)} = \frac{\exp(2z_i) - 1}{\exp(2z_i) + 1}, \tag{2.7}$$

and the output values are between -1 and 1. Furthermore, the hyperbolic tangent can be given in terms of the sigmoid function with $\tanh(z_i) = 2\sigma(2z_i) - 1$. Hyperbolic tangent is an option if the output value has to be bounded and centered at zero, and is often used in so called recurrent neural networks. In addition, some modifications to ReLU have been used. One of these modifications is Leaky ReLU [12], defined by

$$g(z_i) = \begin{cases} 0.01z_i, & z_i < 0 \\ z_i, & z_i \geq 0. \end{cases} \tag{2.8}$$

2.3.3 Convolutional neural networks

For a fully connected layer the output of the layer can be computed as in Eq. 2.3 by using a simple matrix multiplication and vector addition before applying the activation function. Instead of computing the affine transform $\mathbf{Wx} + \mathbf{b}$, in convolutional layers the input is transformed using an operation called convolution. Convolutional neural network (CNN) is a neural network that contains at least one convolutional layer. [10]

The convolution is usually denoted with $*$ and in one-dimensional discrete case it can be written as

$$s(t) = (x * w)(t) = \sum_{i=-\infty}^{\infty} x(i)w(t-i), \tag{2.9}$$

where the x is usually referred to as the input, and w as the kernel of the convolution. In applications the input is an array of data, and the kernel is an array of parameters learned during the training process. Often the arrays are multidimensional and called tensors. Moreover, in practice the kernel and data is assumed to be nonzero over only a finite set of points so that the sum in the convolution has a finite number of terms. [10]

The convolution is commutative, meaning that $(x * w)(t) = (w * x)(t)$. Moreover, the convolution can be taken over several axis at a time. If I is an input image in a two-dimensional array and K is a two-dimensional kernel, we can write

$$S(i,j) = (I * K)(i,j) = \sum_k \sum_l I(k,l)K(i-k, j-l), \tag{2.10}$$

which is possible to turn around to $S(i,j) = (K * I)(i,j)$ by the commutativity. [10]

Often convolutional layers are followed by a pooling function. Pooling acts on a neighborhood of a point, and replaces the output of a single point by some function of its neighborhood. Common choices are taking the maximum value or the average. Pooling helps to make the layer output invariant to small translations in the input. [10]

Figure 2.2 shows a visualization of the data and the changes in dimensionality throughout a convolutional network.

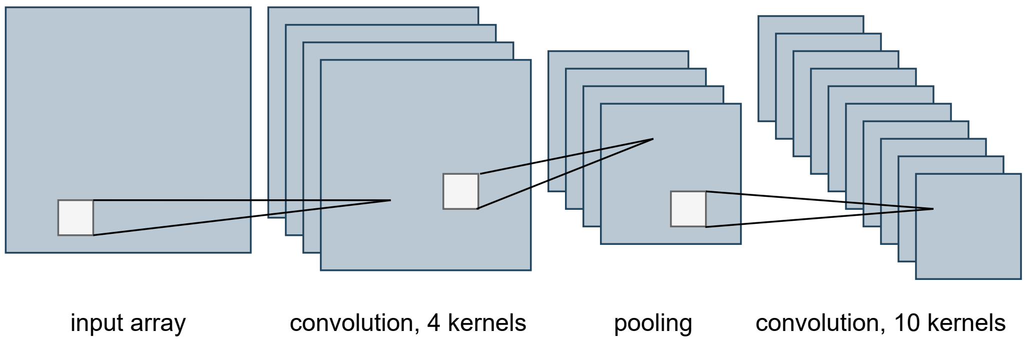

Figure 2.2. *Illustration of the first layers of a convolutional network.*

2.3.4 Training

Training a neural network is a similar task to many other machine learning problems, and it is done by gradient-based learning. The training requires a cost function, and the layer weights are updated according to the gradient. However, the mapping from a network's inputs to its outputs is constructed by composing the layers, and is usually too complex to compute the gradients directly. Nevertheless, the information from the cost function can be fed back to the network with the back-propagation algorithm. [10]

The gradient descent is a simple iterative method to find a local minimum (or maximum) of a differentiable function. The direction of the fastest decrease of the function F in a neighborhood of $\mathbf{x}$ is given by the negative gradient $-\nabla F(\mathbf{x})$. The gradient descent update rule can be then given by

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma \nabla F(\mathbf{x}_n), \tag{2.11}$$

where γ is the learning rate. If γ is sufficiently small, then $F(\mathbf{x}_{n+1}) \leq F(\mathbf{x}_n)$, and eventually the algorithm converges. With well-behaving F and ∇F gradient descent usually performs well, but in case of more complex functions to optimize it may be difficult to find an extreme point. [10]

In modern neural network training often some version of stochastic gradient descent (SGD) algorithm is used. The basic idea behind SGD is that computing the gradient for every input sample is computationally costly, and the gradient is an expectation that can be approximated by using the average gradient of a smaller sample. The minibatch used to estimate the gradient is drawn uniformly from the traing data, and usually its size varies from one to several hundred samples. In addition to deep learning, SGD is useful in learning large linear models when the amount of data is very large. [10]

The basic SGD can be extended in various ways. One of the most common methods is Adam, which uses running averages of the gradients and the element-wise squared gradients to estimate a non-constant learning rate. [13]

By definition deep neural networks contain many consecutive layers. In the gradient based learning the gradient tells how the layer's parameters should be updated, whereas the other parameters are assumed to stay constant. However, in practice all the weights are updated at the same time, and in a deep cascade of layers this may yield unexpected effects [10]. Batch normalization[14] is a method to reduce the effect of coordinated update of weights, and can be applied to any input or hidden layer of the network. In batch normalization the input samples of a minibatch are normalized by

$$h'_i = \frac{h_i - \mu_i}{\sigma_i}, \tag{2.12}$$

where μ_i and σ_i denote the average and standard deviation, respectively. In practice a small constant is added to the standard deviation to avoid dividing by zero. [10]

2.3.5 Autoencoders

An autoencoder (AE) is a neural network consisting of an encoder and a decoder, and in between the two is a bottleneck layer. As the name suggests, the number of units in the bottleneck layer is small compared to the input dimension. The autoencoder is trained using the same output target as the input data, so that it should reconstruct the original input. With this construction the bottleneck layer can be interpreted as a lower dimensional representation of the original input. [15]

The autoencoder is a widely used method for detecting anomalies in the input. For the data with known properties, such as the data that was used for the training, the output of an autoencoder will be close to the input. However, if the input contains some properties unknown to the autoencoder, those will result in large reconstruction error [16]. In other words, an autoencoder learns to reconstruct its training data relatively well, but fails when the data has different characteristics. In many cases the autoencoder is an attractive choice for anomaly detection since the training does not require obtaining data containing the anomalies expected to be detected.

In audio domain a common choice for the autoencoder input is to take several consecutive frames from the signal spectrogram and only employ dense layers [17, 18, 19]. In this approach the resulting errors of moving frames have to be combined in some way. A natural choice is to average over absolute errors or squared errors to obtain mean absolute error (MAE) or mean squared error (MSE), respectively. These can be computed with

$$MAE(\hat{x}) = \frac{1}{N}\sum_{k=1}^{N} |\hat{x}_k - x_k|, \tag{2.13}$$

and

$$MSE(\hat{x}) = \frac{1}{N}\sum_{k=1}^{N}(\hat{x}_k - x_k)^2, \tag{2.14}$$

where $\hat{x}$ is the reconstruction and x the original. For a smaller number of consecutive spectrogram frames the dimensionality of the autoencoder input stays relatively small, but as the length of the input increases it is more convenient to use convolutional layers [20]. Furthermore, even raw audio can be used with an autoencoder [21] by using strided convolutional layers for the input and WaveNet [22] for the reconstruction.

A variational autoencoder (VAE) is a modification of the autoencoder where instead of a lower dimensional representation the low-dimensional latent distribution parameters are learned. The output is then reconstructed by decoding a sample from the latent distribution. Therefore a major difference to an autoencoder is that VAE is a stochastic method, whereas AE is purely deterministic [23].

2.3.6 Siamese networks and few-shot learning

A siamese neural network is a setup where an artificial neural network is trained using several input samples at a time. The same network weights are used for computing feature vectors for all the inputs. These vectors or their distances can then be used for further computation. [24]

A network trained with a siamese setup can be used for dimensionality reduction. The network computes a nonlinear transformation from the input to an embedding space with a predefined number of features. How the inputs are mapped into the embedding space is determined by the training process. For example, a network can be trained to place the input samples from the same class closely together, and as far as possible from samples of different classes. Furthermore, even if there are only a few class examples available, the vectors in embedding space can be still used for few-shot learning. For each class example we can compute its feature representation in the embedding space. A new sample is then classified by computing its representation and using e.g. a nearest neighbor classifier in the embedding space [25].

One possibility to train a siamese network is to use triplet loss. A block diagram of the setup can be seen in Figure 2.3. In the training process every input contains an anchor sample and corresponding positive and negative samples. The positive sample is chosen from the same class as the anchor, and their embeddings should be placed close to each other. The negative sample is from a different class, and its representation in the embedding space should have a margin to the anchor. In the training process the same network is used to compute embeddings for all the inputs, and the triplet loss is calculated

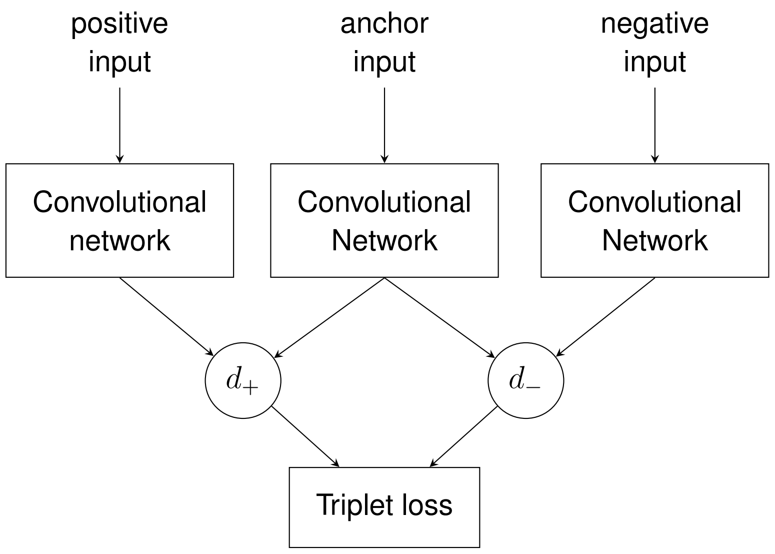

Figure 2.3. *Block diagram of the triplet setup for training a siamese network.*

as

$$L(x_a, x_p, x_n) = [d(f(x_a), f(x_p)) - d(f(x_a), f(x_n)) + \alpha]_+, \tag{2.15}$$

where f is the mapping of the network, x_a, x_p, x_n are the anchor, positive, and negative inputs, respectively, and $d(\cdot,\cdot)$ is a distance function, α is a margin constant that should separate positive and negative classes, and $[x]_+$ denotes $\max(x, 0)$ [25].

Often the output of the embedding network is normalized [25, 26, 27], but also output regularization has been used [28]. Moreover, the results in [29] suggest using output normalization. If the embedding vectors are normalized, a simple relation for distance metrics can be derived. If x and y are vectors in the embedding space, then their ℓ^2 distance is simply

$$d_2(x, y) = \|x - y\|_2. \tag{2.16}$$

The cosine distance is another common metric, which is defined as

$$d_c(x, y) = 1 - S_C(x, y), \tag{2.17}$$

where S_c is the cosine similarity from the inner product, defined as

$$S_c = \frac{\langle x, y\rangle}{\|x\|\|y\|}. \tag{2.18}$$

Now if the embedding vectors are normalized, the cosine distance becomes $d_c(x, y) = 1 - \langle x, y\rangle$, and furthermore

$$\|x - y\|_2^2 = \|x\|^2 + \|y\|^2 - 2\langle x, y\rangle = 2 - 2\langle x, y\rangle = 2d_c(x, y), \tag{2.19}$$

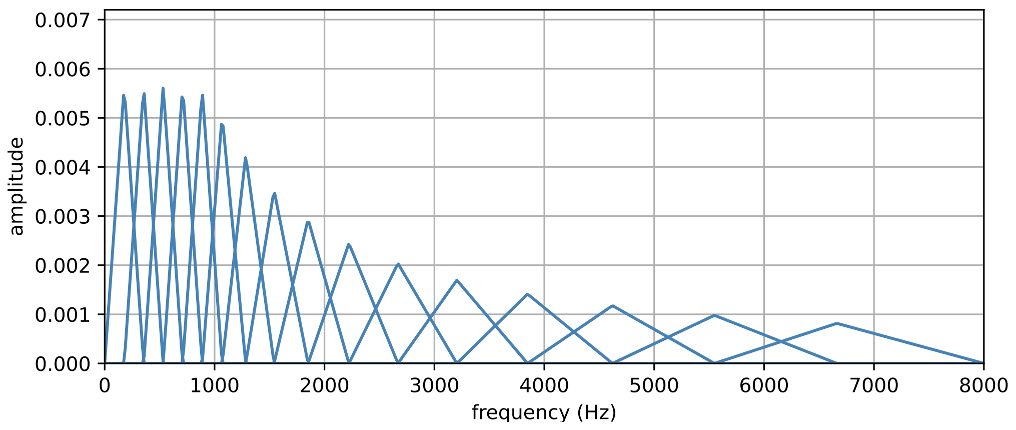

Figure 2.4. *16 bin mel filterbank on a linear frequency scale using sample rate 8 kHz.*

and therefore $d_2(x, y) = \sqrt{2d_c(x, y)}$. Moreover, since the vectors are normalized we have $d_2(x, y) \leq 2$ and $d_c(x, y) \leq d_2(x, y)$. In other words, the cosine distance is always smaller in the embedding, and it is proportional to the square of the Euclidean distance.

2.4 Audio analysis

The most straightforward way to represent audio in a digital format is to store the waveform. However, for the analysis it can be more beneficial to work with the frequency content. There are some network implementations using raw audio, for example [30], but the vast majority of the current research uses some representation of the frequency content as an input.

A spectrogram is a way to represent the frequency content against time. The spectrogram is computed taking FFT of moving fixed length sample windows, and therefore the resulting 2D array dimensions depend directly on the used parameters. The number of frequency bins is half of the length of the FFT window, and how much the window is moved affects directly to the temporal axis resolution. One theoretical difference between the spectrogram and raw waveform representation is that the spectrogram only contains the powers of the frequency components, and the information about their phases is lost.

The frequency scale in an FFT spectrogram is linear, whereas the human perception for pitch is closer to a logarithmic scale [31]. This motivates using of a quasi-logarithmic conversion such as the mel scale for the frequencies to obtain better representation in sense of human hearing.

The usual way to convert an FFT spectrogram into a mel-spectrogram is to use a filterbank. A mel filterbank consists of a given number of triangular filters placed on the linear mel scale. Furthermore, the maximum amplitude of every filter can be normalized by its bandwidth. Figure 2.4 shows a mel filterbank with 16 bins constructed with librosa. Con-

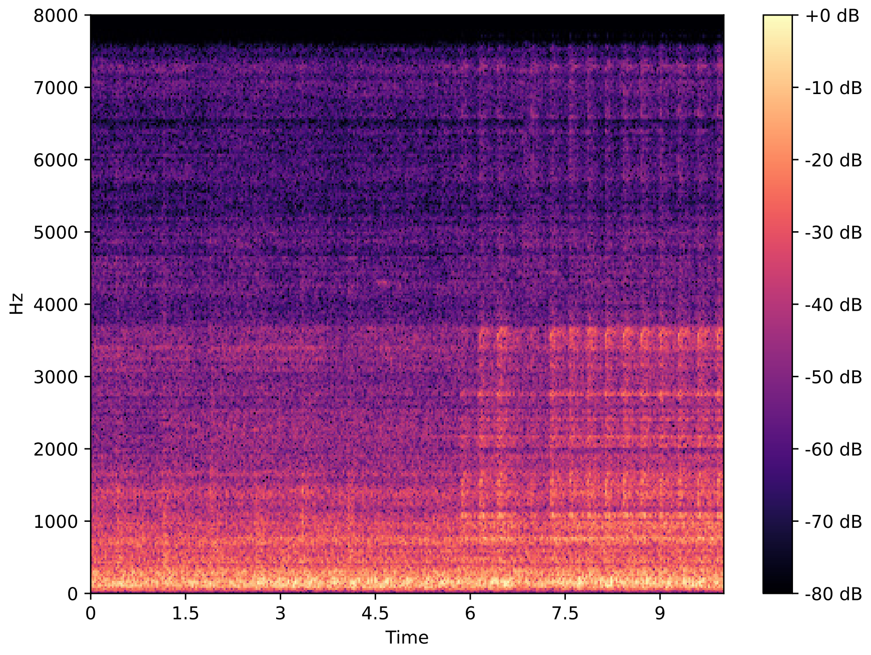

Figure 2.5. *Example spectrogram. The sample rate is 8 kHz and for the spectrogram FFT length 1024 and hop length 512 is used.*

verting one FFT frame into mel energies can be done by multiplying the FFT vector by the filterbank matrix, and converting the whole spectrogram is a single matrix multiplication. An example of a spectrogram can be seen in Figure 2.5, and Figure 2.6 shows the mel spectrogram of the same signal.

There are several different versions of the formula for calculating the mel scale. For example, by default librosa [32] uses the formula

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right). \tag{2.20}$$

2.5 kNN-classification

The k-nearest neighbors (kNN) is a nonparametric classification algorithm. For the kNN classification the training phase is simply storing the training data and their corresponding class labels. An input is then classified by measuring the distances to the training samples, and taking the majority vote for the class of the k closest samples. For k any positive integer can be chosen, the simplest choice being $k = 1$ where the input samples are classified according just by finding the closest training sample. [33]

One downside of kNN is that since the distances are calculated to every known sample,

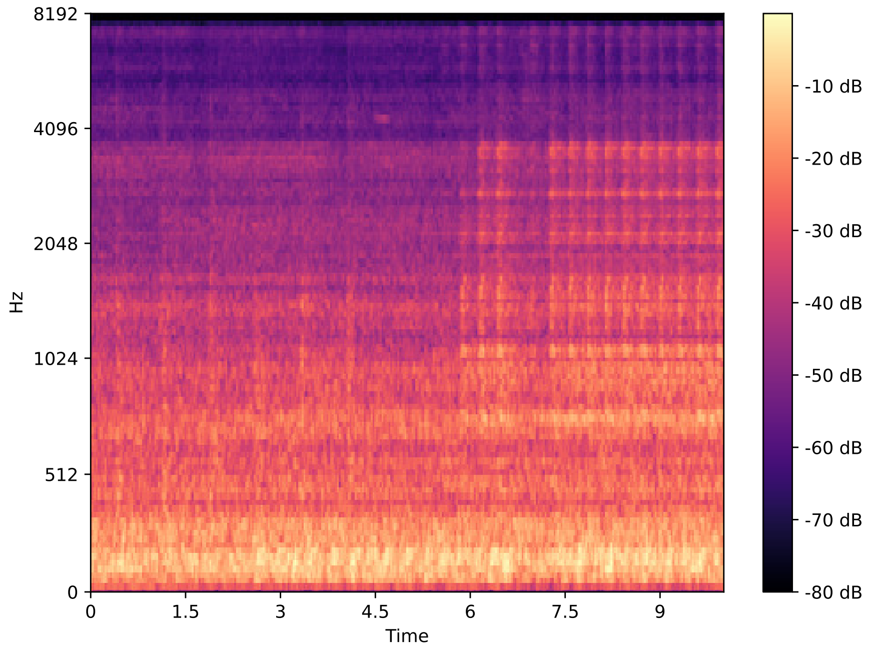

Figure 2.6. *Example mel spectrogram. The sample rate is 8 kHz and for the spectrogram FFT length 1024 and hop length 512 is used, and the number of mel filters is 128. y-scale indicates the frequencies of the mel bins.*

the computational complexity grows with the amount of training data. One way to overcome the problem is to reduce the amount of samples by only using the cluster centers. [33]

3 METHODS

3.1 Datasets

In the experiments two different datasets are used, one containing real recordings, and another one synthetically generated. The real anomaly detection dataset is one of the benchmark datasets provided within the DCASE challenge [34]. The synthetic dataset was constructed by mixing various recorded sound events with background recorded in real industrial site in Finland.

3.1.1 Anomaly detection datasets

The data used in training and testing is the development dataset from DCASE 2020 task 2 [34], and is originally from ToyADMOS (Toy anomaly detection in machine operating sounds) [17] and MIMII (Malfunctioning industrial machine investigation and inspection) [18] datasets. ToyADMOS contains audio from machine-like toys, and during the data collection the toys were damaged on purpose to obtain anomalous sounds [17]. MIMII contains sounds from four different types of real machines: fans, water pumps, slide rails and valves. Anomalous sounds contain various types of abnormal conditions for every class [18].

The data consists of ten-second sound clips from six different machine classes, and every machine class contains several different machines. The class ToyConveyor only has three different machines, whereas the five other classes have four different machines. The machines are recognized with a machine id tag. Figures 3.1 and 3.2 show the mel spectrograms of a normal and anomalous ToyConveyor sample, respectively. Both of the samples are from the device id 01.

The files are single channel and the sample rate is 16 kHz. The training and test sets consist of 20119 and 10868 files, respectively. The numbers of files in individual classes can be seen in Table 3.1. In total the sound data takes 9.7 gigabytes on disk. In the remaining part of this work the dataset is called the DCASE 2020 dataset.

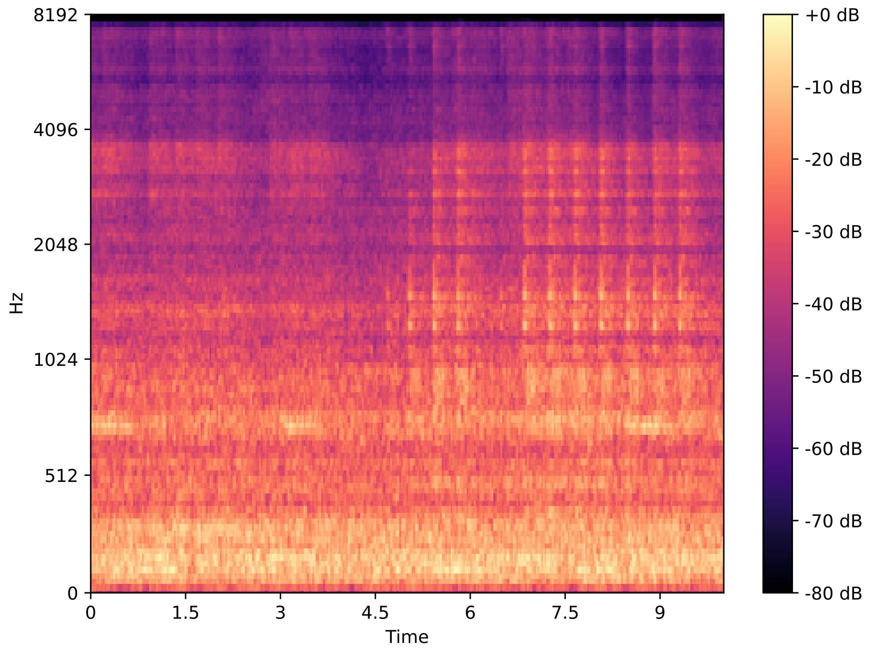

Figure 3.1. *Mel spectrogram of a normal ToyConveyor sound example.*

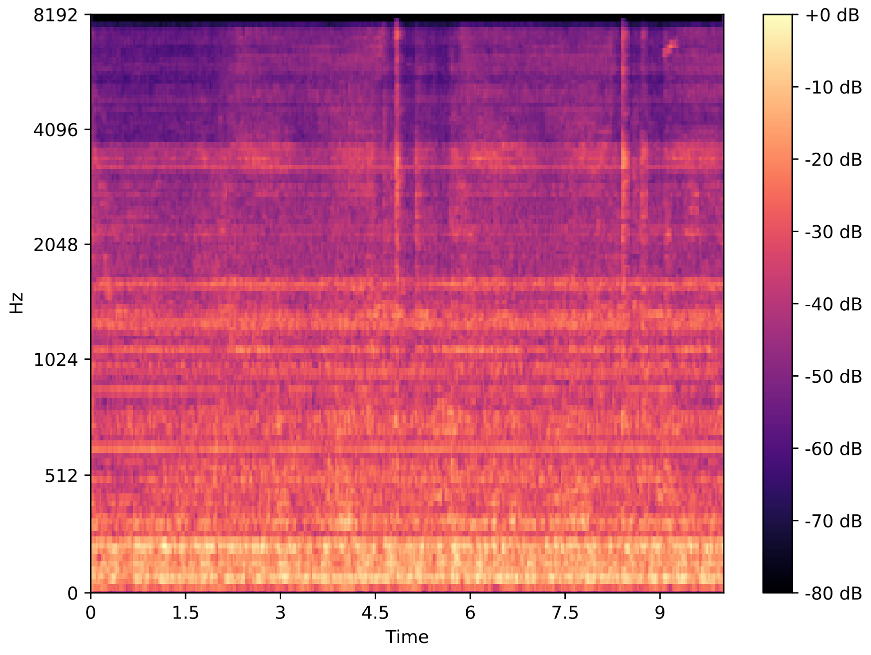

Figure 3.2. *Mel spectrogram of an anomalous ToyConveyor sound example.*

Table 3.1. *Numbers of files in individual classes in the DCASE 2020 dataset.*

	Train	Test	
Machine	Normal	Normal	Anomaly
Fan	3675	400	1475
Pump	3349	400	456
Slider	2804	400	890
ToyCar	4000	1400	1059
ToyConveyor	3000	2399	1110
Valve	3291	400	479

3.1.2 Synthetic data

During the research work an artificial dataset was constructed for development and testing. Real world data recorded by IndMeas was used as normal sounds and background noise for anomalous samples. For the anomalies various samples from freesound.org were collected. These samples included scratches, squeaks, creaks, and breaking of various things to mention a few. In addition, some artificial anomalies were recorded by IndMeas[1]. These samples included some tools, e.g. drill and grinder, and some nearby construction site sounds. The total number of anomalous samples used was 448. Finally, some anomalies were created just by pitch shifting the original sound. For a more detailed study, the anomalies were divided into six different classes: five classes based on recordings by IndMeas, and one class from the FreeSound data.

The background data was recorded continuously, and the data was divided into 15 second clips in the process. Every recorded background sound was used at most once: either as a normal sample, as background for an anomalous sample, or discarded. To ensure that there is no overlapping, 200 samples between the training and test samples were discarded. In terms of time the amount of discarded samples translates to a 50 minute gap between the recordings of training and test sets. Finally, the sounds were processed in lexicographical order, and the test data was generated first. Anomalous samples were mixed on top of the background samples in such a way that a predetermined SNR level was obtained. The synthetic anomalous samples were created for three different SNR levels of -6dB, -10dB, and -20dB.

The constructed training data consists of 3478 samples recorded between 0.00 - 7.30 UTC. The numbers of different anomaly samples can be seen in Table 3.2. The third class labeled as Outdoor noises contains sounds outside the planned recordings, mostly originating from a nearby construction site. The samples in the third class include sounds from trucks, distant talking, clatter and thumping.

[1]https://www.indmeas.com

Table 3.2. *Classes of anomalous samples in the synthetic data.*

Class	Number of files
Drill	132
Drill with hammer	13
Outdoor noises	37
Grinder on idle	118
Grinder	3
FreeSound samples	145

3.2 Two stage system

The framework developed during this work consists of two neural networks. The block diagram of the system is shown in Fig. 3.3. The purpose of the first stage is to discard all the incoming normal (non-anomalous) samples, whereas the second stage is used to classify the anomalous data that passes through the first stage and take the decision of triggering or not an alarm.

The input for the system is a log-mel spectrogram computed from the inspected audio sample. The spectrogram is computed using 1024 sample FFT with 512 sample hop length. For the mel 128 bins are used.

The first stage is responsible for detecting the anomalies, and discarding the input data from normal conditions. This is implemented by using a standard autoencoder. All input samples that are tagged as anomalous are then passed on to the second stage.

In the second stage a convolutional network is used to compute a feature vector of the anomalous input sample in the embedding space, where it is then classified using the nearest neighbor approach. In addition to just looking at the nearest known sample, the distance in the embedding space between the input sample and the nearest neighbor is also taken into account. If the mapping of the input vector is not near any known anomaly class example, it is considered as a novel sample and classified as an alarm.

The two stages are independent in such a way that the implementations of each one can be changed freely. The purpose of the first stage is to detect abnormal sounds, and any anomaly detector available can be used for it. Similarly, the implementation of how the embedding is computed can be changed in the second stage. Finally, instead of using just nearest neighbor some other classification method could be used for the alarm-trigger decision.

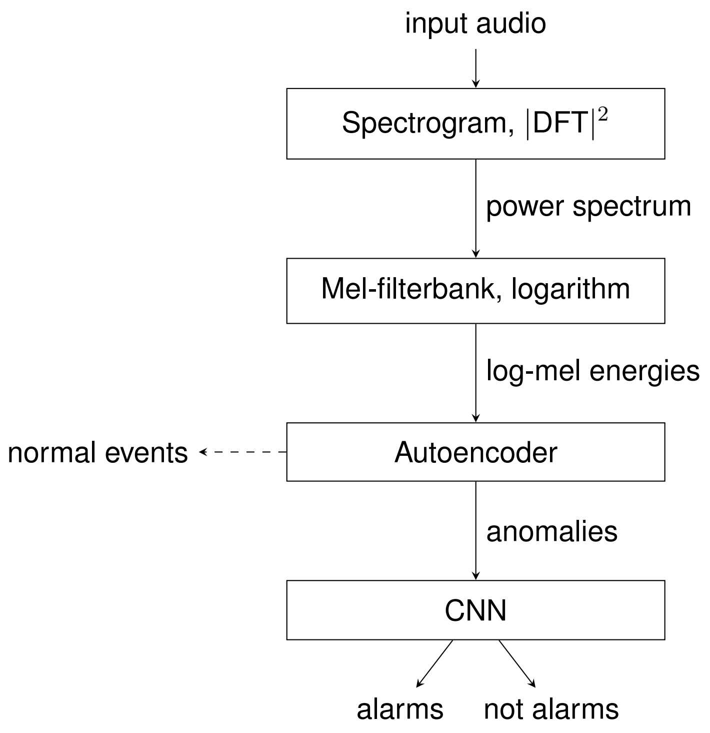

Figure 3.3. *Block diagram of the proposed system.*

3.2.1 Stage 1: Autoencoder

For the autoencoder an architecture similar to the DCASE 2020 baseline was used [34]. The encoder and decoder parts both contain four dense layers of 128 units with ReLU activation, and batch normalization layers are used after every dense layer. The bottleneck layer contains 16 units. The autoencoder input is five consecutive frames of the log-mel spectrogram, and therefore the total input dimension is 640. As the data is processed at 16 kHz sampling rate, 1024 samples FFT window length and 512 samples hop length result in 0.192 second input length.

The anomaly score for the autoencoder is computed using the mean square error (MSE) of the reconstruction and the input. The five frame input window is advanced one frame at a time and MSE for every step is calculated. The final anomaly score is the mean of the reconstruction errors.

Anomaly detection with an autoencoder is done by deciding a threshold value for the anomaly score. If the computed anomaly score for an input exceeds the predetermined value, the sample is tagged as an anomaly.

3.2.2 Stage 2: Convolutional network

The convolutional part of the embedding network consists of three blocks, each having a convolutional layer with 5$\times$5 size filters with ReLU activation. In each block batch normalization and max pooling are used after the convolutional layer. The first block contains 64 filters, the second 96, and the third 128 filters. The size of the filters was chosen to be the same as in the acoustic scene classification network in [35]. After the convolutional blocks the output is flattened and fed into a single dense layer with linear activations. The dense layer is used for constructing the embedding. Finally the output is normalized to have an unit Euclidean norm.

The parameters of the system were chosen in such a way that the input size of the convolutional network corresponds roughly to the ten-second length in the DCASE 2020 dataset files. The input size of the CNN was chosen to be of size 128 $\times$ 304. That is, the input spectrogram should contain 304 frames with 128 log-mel bins. With the 16 kHz sample rate, 1024 sample FFT and 512 sample hop size this corresponds to 9.76 seconds. With the selected settings there are approximately 2.1 million trainable parameters.

The CNN is trained using triplet learning with a margin $\alpha = 0.3$ and Euclidean distance metric. In the implementation the loss function is constructed inside the model, and the model output is the triplet loss. Setting the target values as zeros and using mean absolute error as the loss for the training, the actual training loss becomes the mean of the triplet losses.

Going through every possible triplet is practically impossible, therefore the triplets generated for training are chosen randomly. The process generates two types of triplets. For easy triplets the negative sample is chosen from a completely different machine class. For hard triplets the negative sample is chosen from the same machine class, but different machine id. The positive sample is always chosen from the same machine class and machine id. In the training process every training sample is an anchor sample once in an epoch. Whether a triplet is easy or hard is randomly chosen, with both possibilities having equal probability of 50%.

The anomalous input samples that are passed onto the second stage are mapped into the embedding space using the CNN, and the decision of triggering the alarm is done using a kNN classifier. In this work $k = 1$ and the Euclidean distance is used to measure the distance of two points in the embedding space.

4 RESULTS AND DISCUSSION

4.1 Synthetic data results

The first stage autoencoder was trained and tested with the synthetic data. The performance was measured using the area under curve metric, and the numbers can be seen in Table 4.1. As the autoencoder recognizes all the pitch shifts, the results in Table 4.1 only contain numbers for the mixed audio. The results show that a simple autoencoder performs well with the data. In other words, the handcrafted anomalies are too easy to recognize or out of the context considering the background soundscape.

For more heterogeneous classes the decline in performance starts earlier, whereas the smaller classes are easier to distinguish from the background. This is probably because the classes contain sounds that are highly different from the ones found in the industrial site soundscape. Moreover, the difficulties to recognize the anomalies in the third class (outside noises) are probably due to the fact that most of the samples do not have any particular content except the background noise from the nearby construction site.

The current available selection of abnormal sounds and the additive mixing is simulating rather abrupt noises, which in turn makes them easy to detect. For mimicking a realistic more subtle changes to the acoustic conditions would be required. Possible spectral modifications could be performed on the normal samples depending on the identified target and possible fault. However, handcrafting a more realistic dataset would require more realistic data and is out of scope of this work.

Table 4.1. *Area under curve results for the baseline autoencoder and synthetic data.*

	SNR			
Class	-6 dB	-10 dB	-20 dB	-30 dB
Drill	1.000	1.000	0.258	0.130
Drill with hammer	1.000	1.000	0.918	0.015
Outside noises	0.992	0.475	0.052	0.010
Grinder on idle	1.000	1.000	0.709	0.128
Grinder	1.000	1.000	1.000	0.161
FreeSound	0.967	0.870	0.390	0.227

Table 4.2. *Autoencoder performance with equal weights for anomalies and normal samples for the DCASE 2020 data.*

Machine	True normal	True anomaly	False normal	False anomaly
Fan	57.18 %	63.38 %	36.62 %	42.82 %
Pump	68.63 %	66.57 %	33.43 %	31.37 %
Slider	82.19 %	79.67 %	20.32 %	17.81 %
ToyCar	78.69 %	70.64 %	29.36 %	21.31 %
ToyConveyor	72.47 %	55.42 %	44.58 %	27.53 %
Valve	46.01 %	76.16 %	23.84 %	53.99 %

4.2 DCASE 2020 dataset results

4.2.1 Stage 1: Autoencoder

For the autoencoder testing, 15 % of the test data was used to find an optimal value for the anomaly threshold value, and the remaining 85 % for the model evaluation. In the implementation the threshold value is chosen by finding a value that minimizes the sum of false normals and false anomalies. A sample is considered as a true normal when the system correctly detects a normal sample. Similarly, a true anomaly is the case when the system correctly detects an anomalous input. A false normal occurs when the system indicates that an anomalous sample would be normal, and a false anomaly is the case when a normal sample is detected to be an anomaly.

Due to the asymmetric role of the first stage, the performance of the autoencoder is measured using the whole confusion matrix that counts all the above defined true and false decisions. Normal samples that are classified as anomalous can be still handled correctly in the second stage, but false normal samples are discarded and result in a missed alarm. To overcome this problem, different weights for false anomalies and false normals can be used in the threshold value optimization.

Table 4.2 shows the results for individual classes when equal weights are used in the threshold optimization. The results using weight 1.5 for false normal are shown in Table 4.3. The resulting numbers are averaged over ten test runs. Figure 4.1 shows the effect of the false normal weight on the amount of samples passed on to the second stage.

4.2.2 Stage 2: Convolutional network

To separate the effect of the errors from the first stage to propagate through the entire system, the second stage was tested using only the anomalous test data. This is identical to the case where the first stage functions perfectly. For the testing of the second stage network, every class was tested individually, and the different machine ids were divided

Table 4.3. *Autoencoder performance for DCASE 20202 data using 1.5 times weight for false normals.*

Machine	True normal	True anomaly	False normal	False anomaly
Fan	31.68 %	83.40 %	16.60 %	68.32 %
Pump	60.18 %	71.03 %	28.97 %	39.82 %
Slider	80.38 %	81.61 %	18.39 %	19.62 %
ToyCar	63.25 %	83.41 %	16.59 %	36.75 %
ToyConveyor	37.31 %	84.63 %	15.37 %	62.69 %
Valve	35.76 %	88.58 %	11.42 %	64.24 %

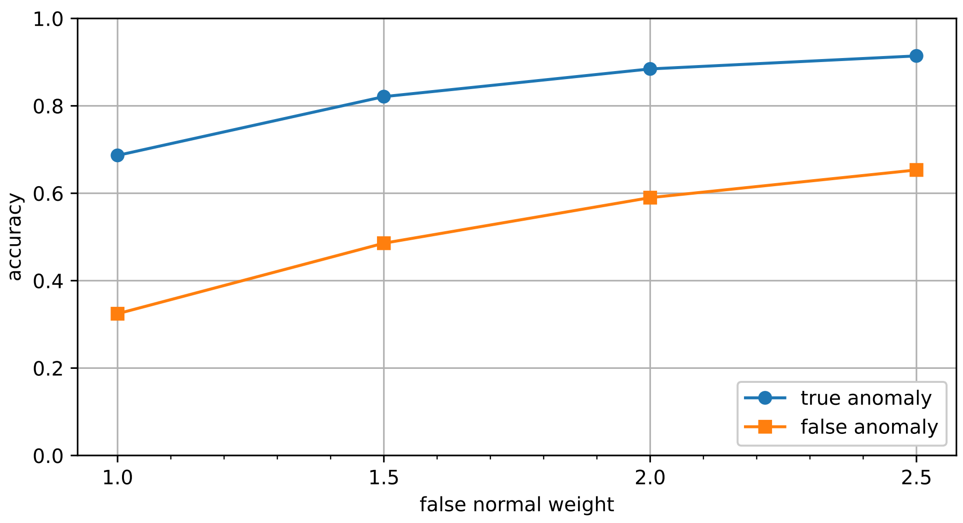

Figure 4.1. *Fraction of anomalies in the first stage as a function of false normal weight.*

into alarm-causing and silenced ones. The silenced classes consist of abnormal sounds but they are from events that should not cause an alarm, such as slamming doors or talking people. 15 % of the test data was used to estimate the distance threshold value used in novel sample detection. If the distance to the closest sample in the kNN is greater than the estimated threshold distance, the input is considered as a novel sample and it triggers an alarm regardless of the kNN classification.

The stage is tested in two setups. First the stage is tested with one-shot classification, where the system has encountered only one example for every class. For the second test the number of known class representatives is increased to five.

The results when only one known example per class is used are listed in Table 4.4, whereas the results for five known representatives per class are listed in Table 4.5. The resulting numbers are averages over ten test runs. For every test run the known class examples are randomly chosen and the distance threshold is optimized.

Figure 4.2 shows the effect of the number of known class representatives on accuracy

Table 4.4. *CNN accuracy for one-shot classification*

Machine	True alarm	True not alarm	False alarm	Missed alarm
Fan	57.61 %	93.13 %	6.88 %	42.39 %
Pump	73.89 %	51.22 %	48.78 %	26.11 %
Slider	98.67 %	50.11 %	49.89 %	1.33 %
ToyCar	45.33 %	71.49 %	28.51 %	54.67 %
ToyConveyor	100.00 %	98.28 %	1.72 %	0.00 %
Valve	95.54 %	93.07 %	6.93 %	4.46 %

Table 4.5. *CNN accuracy for five-shot classification*

Machine	True alarm	True not alarm	False alarm	Missed alarm
Fan	71.97 %	82.75 %	17.25 %	28.03 %
Pump	86.63 %	71.57 %	28.43 %	13.37 %
Slider	99.66 %	76.63 %	23.37 %	0.34 %
ToyCar	71.04 %	61.90 %	38.10 %	28.96 %
ToyConveyor	100.00 %	97.31 %	2.69 %	0.00 %
Valve	100.00 %	90.21 %	9.79 %	0.00 %

for individual classes. The figure shows that increasing the number of known samples improves the performance with ToyCar, pump and slider. The performance with Toy-Conveyor is almost perfect, and there is no significant change in performance when the number of known examples is varied. Adding one more known sample makes the results for fan better, but after that the increase of known class representatives does not affect the average accuracy. However, in the case of fan increasing the number of class representatives evens out the difference between true alarms and silenced ones, and the same is observed for slider. Nevertheless, for pump and valve the effect is the opposite and the difference of the individual class accuracies gets better as the number of known samples is increased.

The average performance over all classes can be seen in Figure 4.3. The figure indicates that even a slight increase from one in the number of known examples per class improves the performance, even with an ideally performing first stage.

4.2.3 Two-stage system

For the complete system testing, 15 % of the data was used for the first stage anomaly threshold optimization, and another 15 % was used for the second stage distance threshold optimization. The 70 % left was then used for the actual model evaluation.

Table 4.6 shows results for the complete two-stage system when the first stage threshold

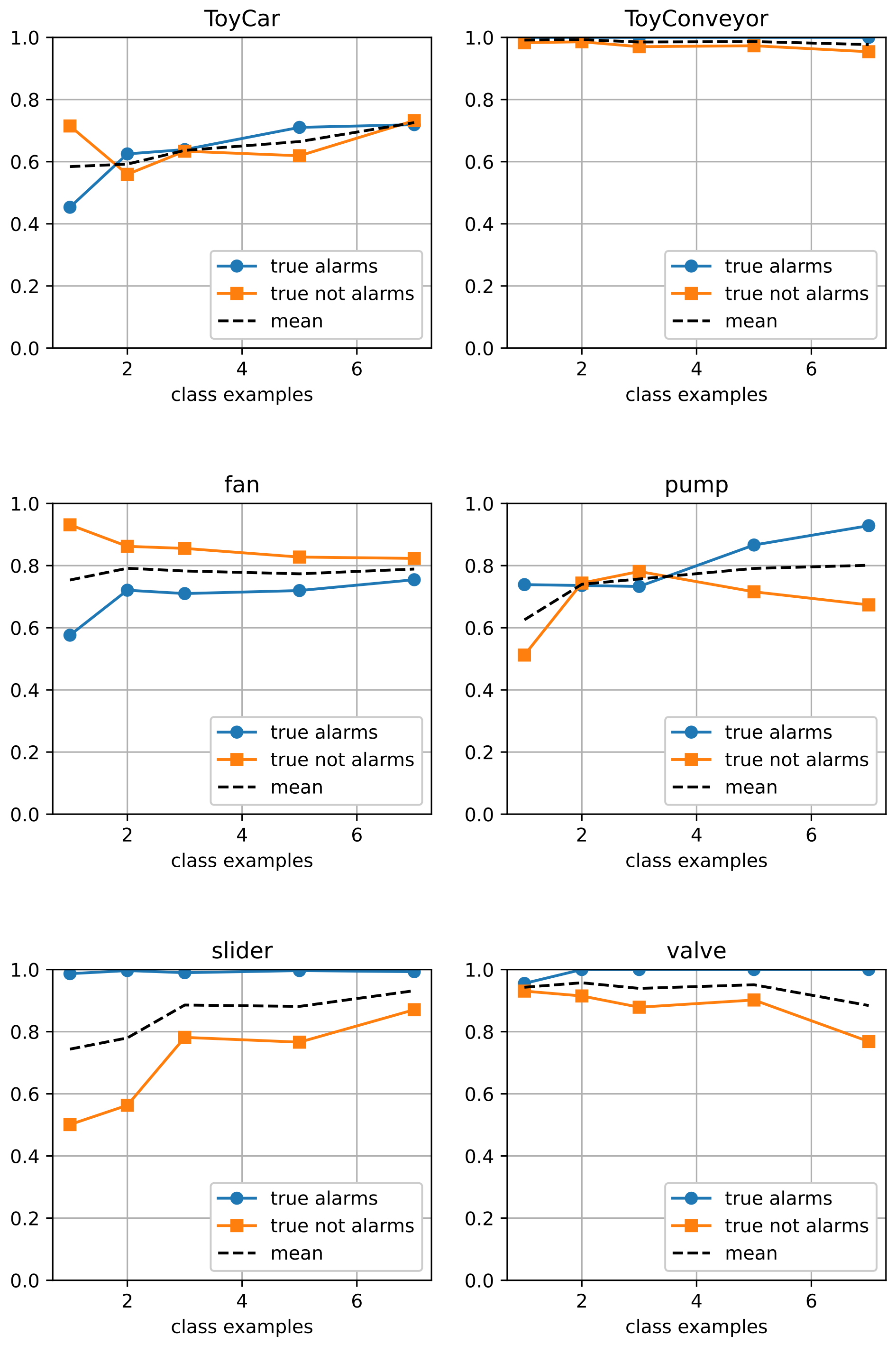

Figure 4.2. *Second stage accuracy for the individual classes.*

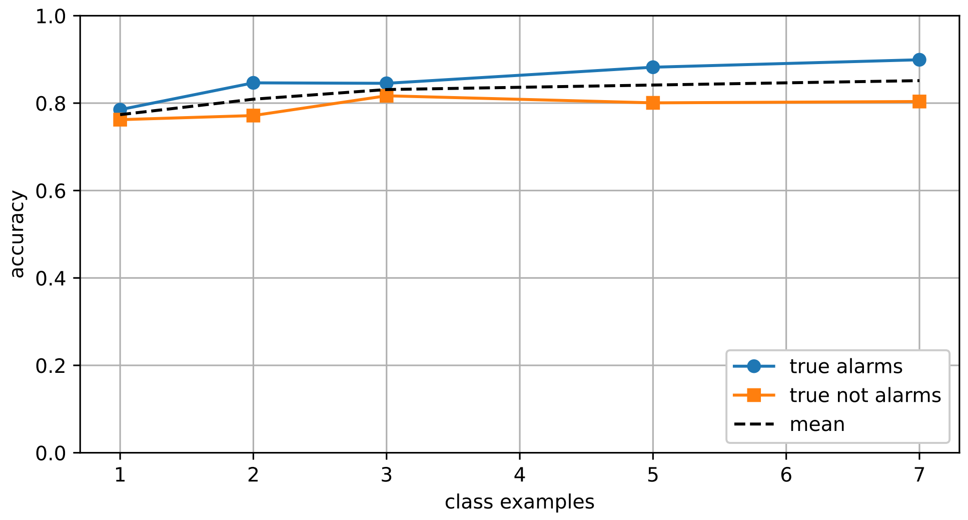

Figure 4.3. *Average accuracy of the second stage as a function of the number of examples per class.*

Table 4.6. *Performance of the two stage system with a 1-shot second stage and double weight for false normal in the first stage.*

Machine	True alarm	True not alarm	False alarm	Missed alarm
Fan	69.56 %	63.57 %	36.43 %	30.44 %
Pump	38.65 %	77.66 %	22.34 %	61.35 %
Slider	51.05 %	83.80 %	16.20 %	48.95 %
ToyCar	52.22 %	80.86 %	19.14 %	47.78 %
ToyConveyor	82.74 %	79.27 %	20.73 %	17.26 %
Valve	91.02 %	73.36 %	26.64 %	8.98 %

is optimized using double weight for false normal and the second stage uses one-shot classification. Similarly, Table 4.7 lists results for the five-shot case. The resulting numbers can be interpreted as frequencies. For example, 80 % true not alarm rate for ToyCar in Table 4.7 means that 4/5 of cases where the input is normal or silenced is not causing an alarm. Similarly, almost 50 % missed alarm rate for ToyCar means that in half of the cases the system is not alarming when it should.

Comparing the results in Tables 4.6 and 4.7 indicate that the machine fan benefits the most of the increase of the number of the known class examples. However, according to the autoencoder results it is also the class that contains the largest amount of false anomalies.

Figure 4.4 shows the accuracy against the number of known class examples for the individual classes. The average over all classes is drawn in Figure 4.5.

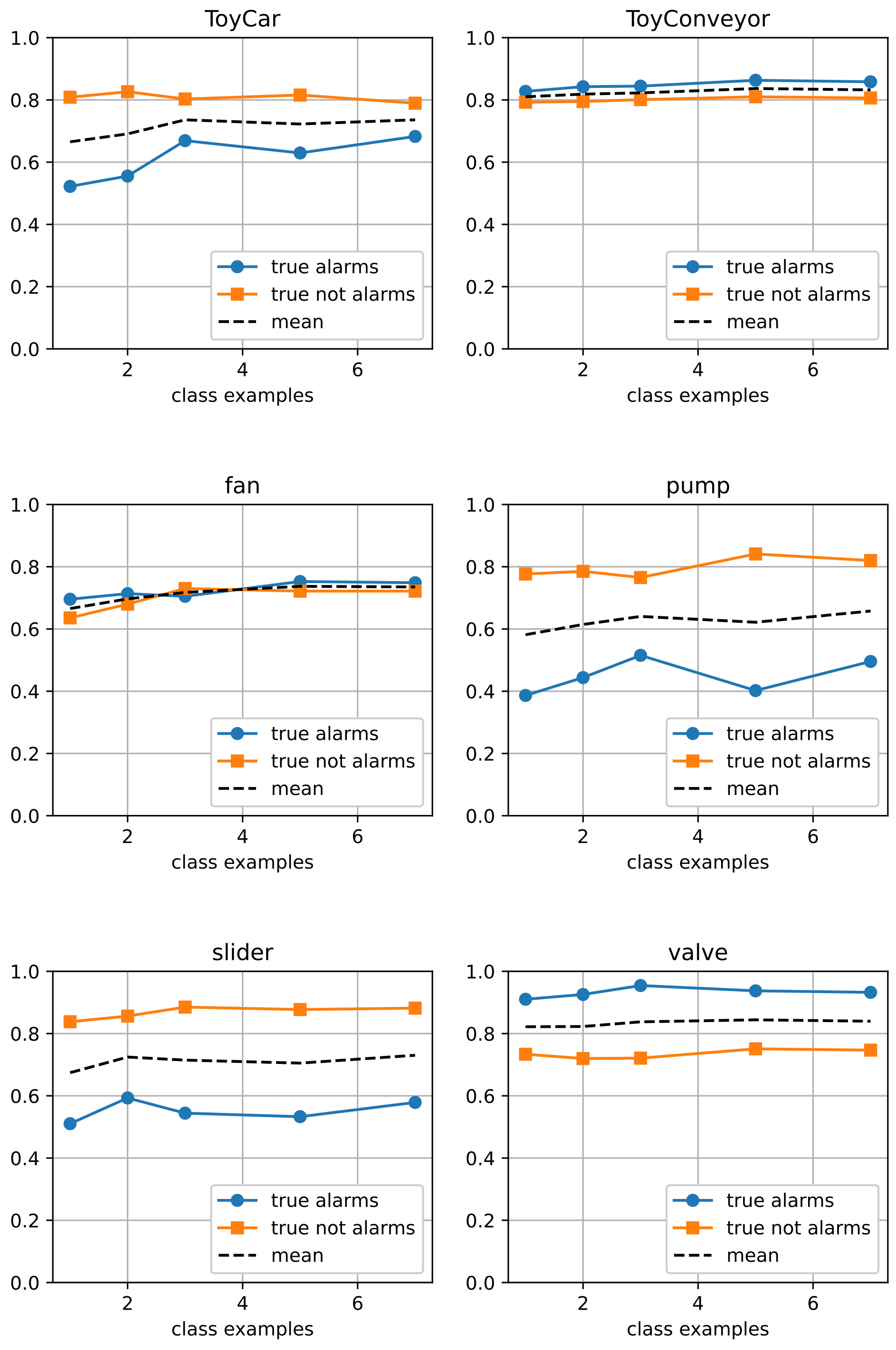

Figure 4.4. *Two stage system accuracy for the individual classes. Double weight for false normal is used in the anomaly threshold optimization.*

Table 4.7. *Performance of the two stage system with a 5-shot second stage and double weight for false normal in the first stage.*

Machine	True alarm	True not alarm	False alarm	Missed alarm
Fan	75.25 %	72.17 %	27.83 %	24.75 %
Pump	40.21 %	84.10 %	15.90 %	59.79 %
Slider	53.29 %	87.72 %	12.28 %	46.71 %
ToyCar	62.96 %	81.57 %	18.43 %	37.04 %
ToyConveyor	86.31 %	81.00 %	19.00 %	13.69 %
Valve	93.73 %	75.09 %	24.91 %	6.27 %

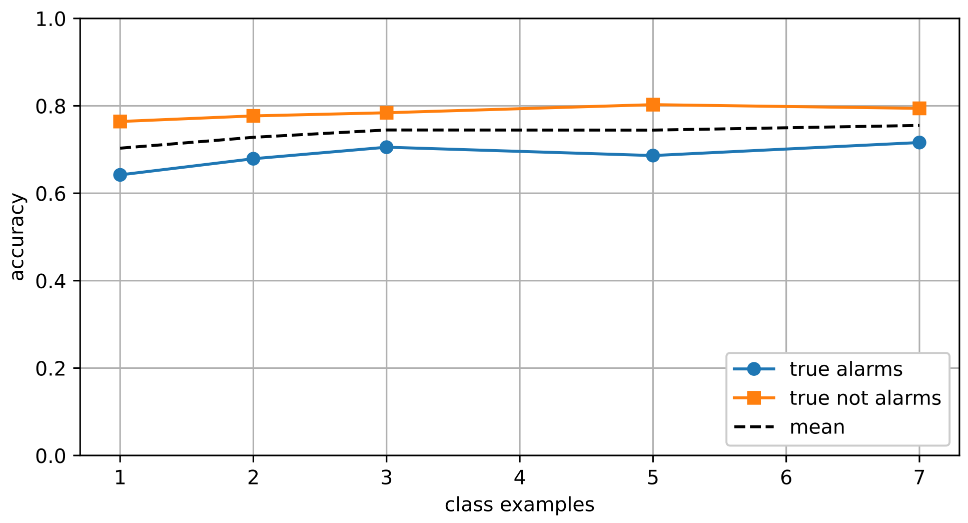

Figure 4.5. *Average accuracy of the two stage system as a function of the number of class examples. Double weight for false normal is used in the anomaly threshold optimization.*

4.3 Discussion and future work

For the testing, different machines inside every machine class are divided into alarm-causing and silenced classes, and therefore the implementation simulates cases where there are two real and two unwanted alarm causes for all the other machines except ToyConveyor. The ToyConveyor class contains three different machines, of which two are silenced and one is alarm-causing in the tests. One known example per class means that the operator has marked an alarm silenced once, whereas five known examples mean that the alarm source was marked as silenced for five times. The results show that increasing the number of known class representatives from 1 improves the average classification results. However, the changes in accuracy vary between machine classes.

These results are obtained in such a way that the system is not learning on the fly. However, in practical implementations it would be beneficial to collect the data during the operation, for example every time when an operator silences an alarm. However, collecting every new sample is not practically possible, and it should be considered in more detail when a new input sample should be stored. Replacing the oldest sample with a new one would give the system ability to adapt to changing conditions. On the other hand, it may not be reasonable to include a new sample if it is very close to some already known sample.

A more critical question is how to choose the threshold values for the system when there are no anomaly examples available. For the first stage it is safer to set lower threshold. If the reconstruction error is larger than the threshold value, the input sample is passed on to the second stage, where it can still be marked as a non-alarm triggering event even if it was incorrectly marked as an anomaly in the first stage. However, for the second stage it is better to pick a threshold high enough, as the classifier works like an ordinary kNN when the threshold is infinite.

Changing implementations in the both stages is possible, and one of the next steps would be to study the effect of using better anomaly detector in the first stage. While changing the implementation of the CNN in the second stage is also possible, the results in Chapter 4.2.2 show that the second stage already performs relatively well.

5 CONCLUSIONS

This thesis proposed an acoustic monitoring system for detecting anomalous events in industrial environments. The choice of having two separate stages was motivated by the application interface to the operator, as the system should allow the possibility to mark unwanted alarms as non-alarm-triggering.

The performance of the stages were studied separately and as a combined system, and the resulting numbers show that the two-stage approach is feasible. For example, for the five-shot second stage and five known class representatives the average true alarm and true non-alarm accuracies were approximately 70 % and 80 %, respectively. In other words, approximately 3/4 events are classified correctly. However, this performance was achieved by using relatively simple implementations in the both stages.

The created synthetic dataset turned out to be too easy, and the anomaly datasets from the DCASE 2020 challenge[34] were used. Creating a useful dataset from the recorded background sounds would require more realistic sounds for the anomalies. However, machine breaks are relatively rare, and recording would require more careful planning.

The two stage framework presented in this thesis confirms the feasibility of an acoustic monitoring system for industrial environments. There is still room for improvement in the resulting numbers, but the proposed approach allows replacing the implementations in the both stages for better performing ones. Setting up the system is a bit cumbersome due to the threshold value choosing, and could be made easier in the future. Finally, studying more sophisticated classification methods instead of the nearest neighbor is left for the future research.

REFERENCES

[1] Vassilopoulos, P. *Models for the Identification of Market Power in Wholesale Electricity Markets*. Tech. rep. Paris Dauphine University, 2003, 46–47.

[2] Forsman, J., Vilén, K., Patronen, J., Revuelta, J. and Ignacio, C. *Selvitystyö tarvittavasta tehoreservin määrästä ajanjaksolle 2017–2022*. Tech. rep. Pöyry Management Consulting Oy, 2016.

[3] IEEE Standard Definitions for Use in Reporting Electric Generating Unit Reliability, Availability, and Productivity. *IEEE Std 762-2006 (Revision of IEEE Std 762-1987)* (2007), 1–75.

[4] Carazas, F. J. G. and Souza, G. F. M. de. Reliability Analysis of Gas Turbine. *Thermal Power Plant Performance Analysis*. Ed. by G. F. M. de Souza. London: Springer London, 2012, 189–220.

[5] Alhanko, A. Energiaraportoinnin kehittäminen voima- ja lämpölaitoksille. Master of Science thesis. Aalto University, 2011.

[6] Kylliäinen, V.-V. and Tiainen-Erkkilä, S. Perussyyanalyysi voimalaitoksessa - Perussyyn etsintä osana ongelmanratkaisua. PSK spring seminar, 2012. URL: `http://vanha.psk-standardisointi.fi/Alasivut/Tiedotteet/Musiikkitalo_2012/5-Perussyyanalyysi%20voimalaitoksessa-Fortum.pdf`.

[7] Paska, J. Reliability and performance indices of power generating units in Poland. *2004 International Conference on Probabilistic Methods Applied to Power Systems*. 2004, 861–866.

[8] Saarinen, J. Voimalaitoksen ennakkohuoltosuunnitelman laatiminen. Bachelor's Thesis. Jyväskylä University of Applied Sciences, 2017.

[9] Schnaare, T. H., Robinson, C. M. and Nelson, R. L. Industrial audio noise monitoring system. 10739187. Aug. 2020.

[10] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. MIT Press, 2016.

[11] Mesaros, A., Heittola, T., Virtanen, T. and Plumbley, M. D. Sound Event Detection: A Tutorial. *IEEE Signal Processing Magazine* 38 (5 2021). DOI: `10.1109/MSP.2021.3090678`.

[12] Maas, A. L., Hannun, A. Y. and Ng, A. Y. Rectifier Nonlinearities Improve Neural Network Acoustic Models. *Proceedings of the 30th International Conference on Machine Learning*. Vol. 28. Atlanta: JMLR.org, 2013.

[13] Kingma, D. P. and Ba, J. L. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)*. 2015.

[14] Ioffe, S. and Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on International Conference on Machine Learning*. Vol. 37. ICML'15. Lille, France: JMLR.org, 2015, 448–456.

[15] Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *Science* 313.5786 (2006), 504–507.

[16] Sakurada, M. and Yairi, T. Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction. *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. MLSDA'14. Gold Coast, Australia QLD, Australia: Association for Computing Machinery, 2014, 4–11.

[17] Koizumi, Y., Saito, S., Uematsu, H., Harada, N. and Imoto, K. ToyADMOS: A Dataset of Miniature-machine Operating Sounds for Anomalous Sound Detection. *Proceedings of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. Nov. 2019, 308–312.

[18] Purohit, H., Tanabe, R., Ichige, T., Endo, T., Nikaido, Y., Suefusa, K. and Kawaguchi, Y. MIMII Dataset: Sound Dataset for Malfunctioning Industrial Machine Investigation and Inspection. *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE2019)*. Nov. 2019, 209–213.

[19] Suefusa, K., Nishida, T., Harsh, P., Tanabe, R., Endo, T. and Kawaguchi, Y. Anomalous Sound Detection Based on Interpolation Deep Neural Network. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, 271–275.

[20] Duman, T. B., Bayram, B. and İnce, G. Acoustic Anomaly Detection Using Convolutional Autoencoders in Industrial Processes. *14th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2019)*. Ed. by F. Martínez Álvarez, A. Troncoso Lora, J. A. Sáez Muñoz, H. Quintián and E. Corchado. Cham: Springer International Publishing, 2020, 432–442.

[21] Chorowski, J., Weiss, R. J., Bengio, S. and Oord, A. van den. Unsupervised Speech Representation Learning Using WaveNet Autoencoders. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* 27.12 (Dec. 2019), 2041–2053.

[22] Oord, A. van den, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. and Kavukcuoglu, K. WaveNet: A Generative Model for Raw Audio. *Arxiv*. 2016.

[23] Kingma, D. P. and Welling, M. Auto-Encoding Variational Bayes. *ICLR*. Ed. by Y. Bengio and Y. LeCun. 2014.

[24] Koch, G., Zemel, R. and Salakhutdinov, R. Siamese Neural Networks for One-shot Image Recognition. 2015.

[25] Schroff, F., Kalenichenko, D. and Philbin, J. FaceNet: A unified embedding for face recognition and clustering. *CVPR*. IEEE Computer Society, 2015, 815–823. ISBN: 978-1-4673-6964-0.

[26] Song, H. O., Jegelka, S., Rathod, V. and Murphy, K. Deep Metric Learning via Facility Location. *International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[27] Law, M. T., Urtasun, R. and Zemel, R. S. Deep Spectral Clustering Learning. *Proceedings of the 34th International Conference on Machine Learning*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, 1985–1994.

[28] Sohn, K. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon and R. Garnett. Vol. 29. Curran Associates, Inc., 2016.

[29] Weinberger, K. Q. and Saul, L. K. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *JMLR* (2009), 207–244.

[30] Ravanelli, M. and Bengio, Y. Speaker Recognition from Raw Waveform with SincNet. *2018 IEEE Spoken Language Technology Workshop (SLT)*. 2018, 1021–1028.

[31] Stevens, S. S., Volkmann, J. and Newman, E. B. A Scale for the Measurement of the Psychological Magnitude Pitch. *The Journal of the Acoustical Society of America* 8.3 (1937), 185–190. DOI: `10.1121/1.1915893`. eprint: `https://doi.org/10.1121/1.1915893`. URL: `https://doi.org/10.1121/1.1915893`.

[32] McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M., Battenberg, E. and Nieto, O. librosa: Audio and music signal analysis in python. *Proceedings of the 14th python in science conference*. Vol. 8. 2015.

[33] Duda, R. O., Hart, P. E. and Stork, D. G. *Pattern Classification*. 2nd ed. Wiley, 2012.

[34] Koizumi, Y., Kawaguchi, Y., Imoto, K., Nakamura, T., Nikaido, Y., Tanabe, R., Purohit, H., Suefusa, K., Endo, T., Yasuda, M. and Harada, N. Description and Discussion on DCASE2020 Challenge Task 2: Unsupervised Anomalous Sound Detection for Machine Condition Monitoring. June 2020, 1–4.

[35] Valenti, M., Squartini, S., Diment, A., Parascandolo, G. and Virtanen, T. A convolutional neural network approach for acoustic scene classification. *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, 1547–1554.