B.Sc.Eng. Thesis
Bachelor of Science in Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

Learning latent audio representations for search and retrieval using supervised and unsupervised learning methods

B.Sc.Eng. in Artificial Intelligence & Data Science

Magnus Guldberg Pedersen
(s204075)

Kongens Lyngby 2024

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

The increasing amount of data created, uploaded, downloaded, and stored every second makes it hard to navigate and find useful data. Audio signals are complex and delicate by nature, making it hard to represent exactly the most important features. This project aims to work towards representing audio signals in a meaningful way.

Motivated by a use case provided by WSAudiology (WSA), this project explores the application of using low-dimensional audio feature representations in a latent space for similarity-based search and retrieval. The project proposes two methods of generating audio feature vector representations: a supervised transfer learning method based on the large-scale convolutional model, Yet Another Mobile Network (YAMNet), and an unsupervised learning method based on a Variational Autoencoder (VAE). The latter is trained on and both are evaluated on the Environmental Sound Classification 50 (ESC-50) dataset, a dataset used for benchmarking environmental sound classification.

The results indicate that YAMNet significantly outperforms the VAE with an accuracy of 91.3% on superclasses in ESC-50 and 83.8% on subclasses compared to 20.5% and 2.6% respectively. The difference in performance is backed statistically by a McNemar test. This difference can most likely be attributed to the fact that YAMNet is used as a transfer learning model and has seen much more data during training as well as hyperparameter tuning and domain specific modeling. The findings of this project contribute to the field of audio feature learning and processing and shed light on the possibilities and pitfalls when using feature representation vectors for similarity search. Suggested future research includes using techniques from YAMNet in the VAE as well as increasing temporal awareness and using a more complicated prior in the VAE model, which may further enhance the performance of the unsupervised approach.

Preface

This project was prepared at the Department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for a Bachelor Thesis on the B.Sc.Eng. program: Artificial Intelligence & Data.

All code developed and used as a part of this project is publicly available at: https://github.com/magnusgp/LLAR.

Kongens Lyngby, December 4, 2024

Magnus Guldberg Pedersen
(s204075)

Acknowledgements

I would like to acknowledge my deep gratitude towards all those who have contributed to the completion of this project. Without the guidance, assistance, and encouragement of some very skilled people, this would not have been possible.

First and foremost, I would like to thank my internal supervisors Bjørn Sand Jensen (bjje@dtu.dk) and Michael Riis Andersen (miri@dtu.dk) for their invaluable support and guidance. Along with their expertise in the field and their excellent feedback, they have helped in shaping the direction and the quality of this project.

I would also like to extend my gratitude towards my external supervisors Jens Brehm Bagger Nielsen (jens.nielsen@wsa.com) and Robert Scholes Lyck (robert.lyck@wsa.com) from WSAudiology who generously shared their time and insight in this field as well as providing the concrete use case for the project.

Contents

CHAPTER 1

Introduction

1.1 Motivation

In a digital age where previously unseen measures of data are created to traverse the internet every single second, finding the relevant information that one really needs can be a discouraging task. Audio signal representations are widely used in various applications and the most obvious being speech recognition and music recommendation, where the latter builds on similarity searching in audio. This search is crucial in other audio-related fields as well, for example when one wants to find similar audio files in a large database and a search structure does not exist. Accurate search will also limit the amount of new audio recordings for databases, which can otherwise be very time-consuming and very costly to acquire. Using a vector representation of the audio features opens up further possibilities to save local space and processing time, two very powerful factors when building and scaling a machine learning service.

With Artificial Neural Networks (ANNs) becoming fast and powerful enough for complex data pattern recognition, audio classification tasks have been carried out by large-scale supervised learning, for example in Convolutional Neural Networks (CNNS) [1]. The supervised nature of these methods meets a natural limit in their performance when large labeled datasets are not available or when the task isn't well-defined. Nonetheless, these methods yield great results when trained in a discriminative fashion on a large labeled dataset.

The introduction of Autoencoders (AEs) and Variational Autoencoders (VAEs) [2, 3] makes it possible to solve some of the issues related to an unlabeled dataset by utilizing unsupervised learning. This project aims to investigate a VAE model for audio feature learning and use the vector representations to find similar representations via search in a larger audio data library using these learned features. This method is then compared to a large convolutional transfer learning model, `Yet Another Mobile Network` (YAMNet), which was trained for audio classification.

1.1.1 WSAudiology

This project is motivated by a use case provided by WSAudiology (WSA). WSA are world-leading in the hearing aid industry. Combining design and cutting-edge tech-

nology, WSA aids several million people with hearing issues. Born out of a fusion between two large hearing aids manufacturers, WSA are experts in audio-related tasks and provides innovative and accessible solutions for hearing-impaired individuals.

WSA has expressed that they are facing issues in audio-related tasks, where it sometimes can be challenging to find the right audio data for a certain task. While having collected many hours of audio recordings, they still find it hard to specifically query a data system for relevant audio recordings. This project seeks to explain some of the processes going into representing audio data in a meaningful way for querying along with investigating Machine Learning (ML) models for this exact purpose.

1.2 Purpose and scope

The project aims to delve into the application of representing features in audio files in a latent space with the representations later utilized for the search and retrieval of similar audio segments. The purpose of this project is to investigate the applicability of using audio vector representations for the search and retrieval of relevant audio. These representations are generated using a supervised Convolutional Neural Network (CNN) model, YAMNet, and an unsupervised VAE. The proposed approach aims to demonstrate its accuracy (wrt. information retrieval) in the audio-related searching task and will hopefully contribute to the field of audio information processing. To do this, this project will assess the following two research questions:

1. In terms of information retrieval performance, how well will an audio vector representation-based method perform when searching for audio by example?

2. Will vector representations of audio files yield better information retrieval when generated by an unsupervised variational autoencoder rather than when generated by a model that has been trained based on a supervised learning method?

This project also explores state-of-the-art methods of variable audio feature learning as well as applicable audio data preprocessing.

The supervised learning method will be based on a large pre-trained convolutional model, YAMNet [4]. An unsupervised feature learning system based on Variational Autoencoders (VAEs) will be achieved by building the Deep Neural Network (DNN) that a VAE is based on and examining the total loss from the model reconstructions. These learning methods will be evaluated and discussed based on relevant information retrieval metrics. Both methods will be evaluated on a widely used dataset for environmental sound tasks, namely the Environmental Sound Classification 50 (ESC-50) dataset [5]. This dataset is a collection of 2000 audio recordings of environmental sounds, which has been used for benchmarking classification methods for environmental audio [6, 7, 8, 9].

1.3 Related work

As the opportunities for improving audio-related work using search in a latent space are many, a large number of technologies have risen. Audio-related work would typically include speech & speaker recognition, speech transformation, recommendation systems of music, source separation, noise reduction, digital audio effects, audio synthesis, and audio classification [10, 11].

Many tools for such tasks exist, however in the scope of using ML for the purpose, these tasks were originally explored with feed-forward neural networks [12] but have later been overtaken by DNNs for these tasks [13]. Speech & speaker recognition has various applications in speech-to-text, voice assistants, and transcription among many others. Audio classification techniques involve labeling audio data related to its origin. Typically, audio classification models are based upon several large data collections of environmental sounds, with the most notable being ESC-50 [5], UrbanSound8K [14] and AudioSet [15]. This is typically done using large-scale CNNs [1] or Long Short-Term Memory (LSTM) models [16] or even a combination of both [17, 18].

Music recommendation systems extend audio data processing into finding similar music/audio pieces. These similar songs are then recommended to the user based on for example their listening history, constantly updating based on user feedback. An example of an application of these systems can be found at `Spotify`, a large streaming service, where similarities are found in a space of vector representations of the music and then used to recommend similar representations to the user [19].

This project seeks to use ideas from both audio classification and music recommendation systems. The general idea is to somehow learn feature representation vectors from audio recordings and then find similar recordings in the space of representation vectors. A popular way of using raw audio data to learn a representation vector is through the `wav2vec 2.0` [20] framework, built upon the language model named `word2vec 2.0`. Other approaches, such as YAMNet include large-scale CNNS, which uses image representations of the raw audio data [1].

Another technique that has seen increased popularity very recently is vector databases. These vector databases utilize vector embeddings from an input such as text, images, or audio and then store the embedding in a latent space where a similarity search can be performed to find vectors close in the latent space. While vector databases use several different embedding algorithms depending on the input data type, they all have in common that they use similarity searching for finding relevant queries in a larger database. Popular instances of large-scale vector databases include Facebook AI Similarity Search (FAISS) [21], Pinecone.io [22] and Milvus.io [23].

Generating vector embeddings with a VAE model is also a popular method for capturing features in data. Current state-of-the-art includes Vector-Quantized Variational

Autoencoder 2 (VQ-VAE-2) [24], which uses a hierarchical VQ-VAE combined with a self-attentive autoregressive model. The VQ-VAE maps the vectors into a finite set of code vectors in a similar fashion to the K Nearest Neighbours (KNN) algorithm.

Since audio data is originally a time series, some methods also explore temporarily aware versions of the VAE. Temporal Difference Variational Autoencoder (TD-VAE) [25] works with sequential data and uses state-space models to generate vector embeddings.

1.4 Outline

Besides the **introduction** in this chapter, the general outline of the report follows this structure: First, a **methodology** chapter outlines the relevant methods and models used as well as practical notes on implementation. Then, the chapter on **experiments** will describe how the research questions described above are going to be answered. The outcome of this is presented in **results & analysis**, where it is also briefly commented on. The results are further elaborated in the **discussion** section. Lastly, the project presents a **conclusion** based on the findings and the research questions.

CHAPTER 2

Methodology

This chapter serves to provide the theory and explanations behind the proposed methods introduced in the introduction. First, the dataset used for this project is described in depth along with preprocessing work. Then, an overview of how the model will be assessed is provided. This includes descriptions of the Information Retrieval (IR) metrics as well as the potential strengths and weaknesses of each metric along with a McNemar test that is used for the comparison of the two learning methods.

These two learning methods are then described in-depth in the last two sections with emphasis on how they generate the audio vector representations used for the experimental setup in this project. Since the VAE is conceptually more complicated than YAMNet, a longer section is provided with derivations of the objective function and assumptions hereof.

2.1 Data

The following section describes the processing of the audio recordings that have been used for the modeling in this project. Both methods and their respective pipelines use the same data as input in order to ensure consistency and a fair evaluation of the IR of the methods. The data is initially the raw waveforms in the audio recordings, but it is transformed into spectrograms during preprocessing in order to fit the input layer of the model architectures proposed in this project.

As mentioned, this project is based on audio recordings from the ESC-50 dataset. These recordings are organized into 5 major semantic categories, each with 10 semantic subcategories. The categories are described in table 2.1.

Animals	Natural soundscapes & water sounds	Human, non-speech sounds	Interior/domestic sounds	Exterior/urban noises
Dog	Rain	Crying baby	Door knock	Helicopter
Rooster	Sea waves	Sneezing	Mouse click	Chainsaw
Pig	Crackling fire	Clapping	Keyboard typing	Siren
Cow	Crickets	Breathing	Door, wood creaks	Car horn
Frog	Chirping birds	Coughing	Can opening	Engine
Cat	Water drops	Footsteps	Washing machine	Train
Hen	Wind	Laughing	Vacuum cleaner	Church bells
Insects (flying)	Pouring water	Brushing teeth	Clock alarm	Airplane
Sheep	Toilet flush	Snoring	Clock tick	Fireworks
Crow	Thunderstorm	Drinking, sipping	Glass breaking	Hand saw

Table 2.1: The semantic super- and subclasses present in the ESC-50 dataset. Each semantic subclass contains 40 unique audio recordings.

Each semantic subcategory consists of 40 unique audio recordings, which all are stored in the Waveform Audio File (.WAV) format. Each recording is a 5-second-long clip sampled at 44.1 kHz, single channel. Five sample data files are shown as wave diagrams in figure 2.1.

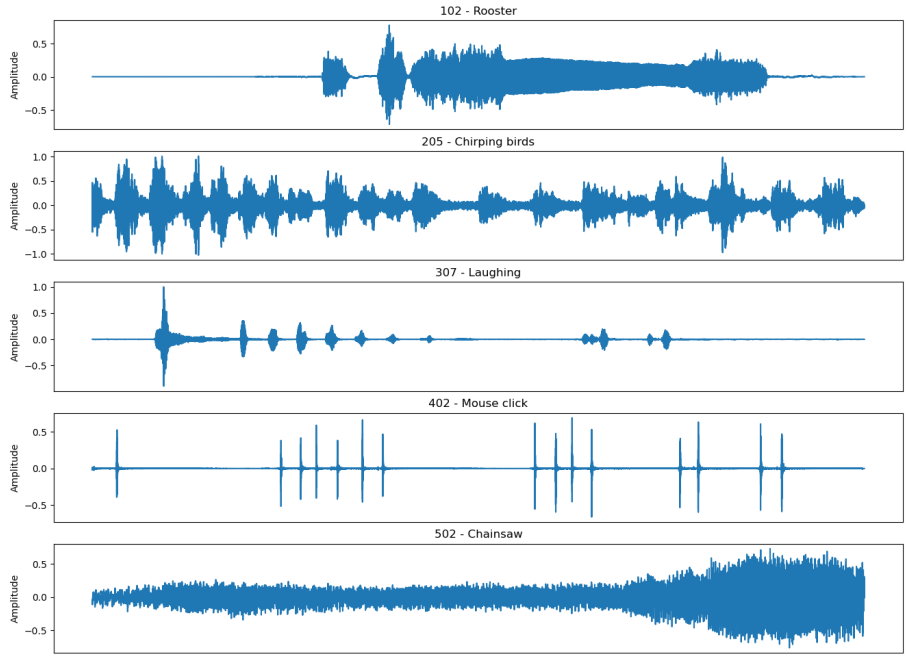

Figure 2.1: One example wave diagram sampled from each of the different semantic superclasses. The length of all recordings are 5 seconds..

The audio recordings were collected from the collaborative online audio-sharing platform, Freesound.org [26]. The dataset follows an original naming convention, which has been modified for this project. Details on this can be found in appendix A along-

side a description of the accompanying dataset meta-data for the dataset including human classification attempts on the data.

2.1.1 Preprocessing

Each audio recording is loaded and resampled into 16 kHz and mono using the python library `librosa` [27]. Then, each file is zero-padded (if needed) to ensure consistency in the number of spectrogram frames that each audio file generates. The Log Mel Spectrograms are then generated for each input waveform. The Log Mel Spectrogram features are scaled by min-max normalization:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{2.1}$$

in order to obtain values in a normalized range of $[0, 1]$. These features are then saved as the final spectrogram, which is ready to be used as input in the models.

The preprocessing steps for the proposed learning methods follow this pipeline:

1. Load audio recordings with samples as 16-bit integers.
2. Normalize the samples to $[0, 1]$ using min-max normalization.
3. Convert samples to mono by taking the mean of the stereo sound (if it is stereo, otherwise nothing happens).
4. Resample audio recordings from 44.1 kHz to 16 kHz.
5. Pad the signal if necessary. Padding is used when there are not enough samples to generate an entire patch window in the spectrograms and when there are samples enough for another patch. In the latter case, the number of patches is rounded up and the signal is padded accordingly.
6. Convert resampled and normalized audio waveform to log Mel spectrogram as described in section 2.1.2.

2.1.2 Log Mel Spectrograms

In order to create input suitable for a convolutional model architecture, each audio signal is converted into a spectrogram by Short Time Fourier Transformation (STFT) [28, 29]. These spectrograms capture the frequencies present in the raw signal and can thereby be used for feature learning. This is done by applying a Discrete Fourier Transformation (DFT) over short overlapping windows on the audio signal to obtain the frequencies present in the signal. The DFT, described as

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi \frac{kn}{N}} \tag{2.2}$$

maps a signal x_n with length N into a frequency domain X_k with N coefficients. By splitting the signal into windows and applying the discrete Fourier transform on the windows w_n, we can extend equation 2.2 to

$$\mathbf{STFT}_{x_n}(h,k) = X(h,k) = \sum_{n=0}^{N-1} x_n w_n e^{-i2\pi \frac{kn}{N}} \tag{2.3}$$

in order to obtain the frequency spectrum for the audio signals.

These frequencies are then transformed into a spectrogram by using filters to obtain the Mel spectrogram. A Mel spectrogram is based on the Mel frequency scale, which represents the way humans perceive pitch rather than the physical frequency scale. This frequency scale emphasizes the lower frequencies in the signal, which makes the sound features closer to what the human ear perceives. To convert a frequency from Hz to mels using the 10 base logarithm:

$$m = 2595 \log_{10}(1 + \frac{f}{700}) \tag{2.4}$$

An example of the transformation from waveform in the recording `101 - Dog/1-30344-A.wav` to Log Mel Spectrogram is shown in figure 2.2.

2.2 Model Assessment

An important part of machine learning is to be able to back the results with certain metrics. IR covers terms that can help measure how much information a certain query gives in a search, which in this case is performed on the latent representations of the audio input. Here, IR can assist in measuring the quality of the learned latent representation and hereby provide insights into how well the supervised and unsupervised learning methods have learned important features from the data.

2.2.1 Information Retrieval Metrics

The metrics are given a set of predicted labels from the search and a set of true labels and will measure how well these match. The metrics are based on reflections in Manning, Christopher D. et al: `Introduction to Information Retrieval` [30]. The IR terms will conclude research question 1 regarding how well the audio vector representation can be used for search and retrieval in other representations.

2.2.1.1 Accuracy

The fraction of correctly classified instances divided by the number of total instances:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.5}$$

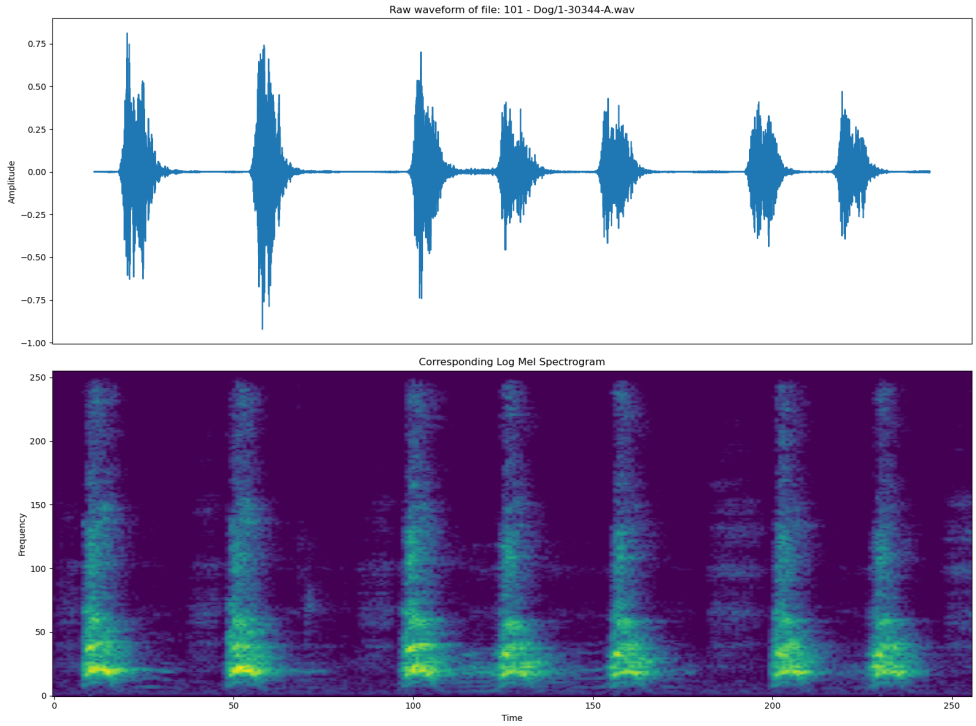

Figure 2.2: Example audio file `101 - Dog/1-30344-A.wav` and its corresponding Log Mel Spectrogram.

where TP is the number of true positives, TN is true negatives, FP is false positives, and FN is false negatives. Accuracy is a very intuitive measure that works well when classes are balanced as is the case in the ESC-50 dataset.

2.2.1.2 Precision & Recall

Precision measures the fraction of true positives over the total number of predicted positives:

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

Recall measures the fraction of true positives over the total number of actual positives:

$$Recall = \frac{TP}{TP + FN} \tag{2.7}$$

These metrics provide more nuanced insight into model performance when compared to accuracy, but are not as easy to interpret.

2.2.1.3 F1-Score

F1-Score is a combination of precision and recall that balances both measures. It is the harmonic mean of precision and recall:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.8}$$

As the F-1 score combines both precision and recall, it is a strong single value that can summarize model performance.

2.2.1.4 Mean Average Precision

Mean Average Precision (MAP) is a metric used to evaluate ranking problems, such as an information retrieval query. It measures the average precision over a set of queries:

$$MAP = \frac{1}{N} \sum_{i=1}^{N} \frac{\sum_{j=1}^{|S_i|} \text{Precision @ } j}{|S_i|} \tag{2.9}$$

where N is the total number of queries, S_i is the set of relevant items for query i, and Precision @ j is the precision at the j-th item in the ranked list. MAP can provide important information regarding how often the model makes a correct classification when considering S_i items. However, this term tends to be high (towards 1) when N is small, it is often useful to combine the MAP with the Accuracy @ K.

2.2.1.5 Accuracy @ K

Accuracy @ K measures the number of true predictions within the top K predictions:

$$\text{Accuracy @ } K = \frac{1}{N} \sum_{i=1}^{N} I(y_{i,k} = t_i) \tag{2.10}$$

where N is the total number of audio embeddings to be searched, $y_{i,k}$ is the predicted label for the k-th prediction and t_i is the true label for the searched representation i. $I()$ is an indicator function that returns 1 if the condition inside is true and 0 otherwise.

Accuracy @ K is a good metric to use when evaluating the top-K predictions of an algorithm. However, it has its limitations if K is not chosen properly, as it does not take any other predictions into account than the top-K ones.

2.2.2 McNemar's test

When considering two models and their accuracy in a given task, it is of interest to measure if these models have significantly different accuracies. The null hypothesis

and the alternative hypothesis for the comparison here would be:

$$H_0 = p(\text{VAE}) = p(\text{YAMNet})$$
$$H_1 = p(\text{VAE}) \neq p(\text{YAMNet})$$

where p(VAE) refers to the probability that the searching experiment with the VAE model retrieves the correct label. The same concept applies to p(YAMNet) for the YAMNet model.

A way to test if the null hypothesis holds is by carrying out a McNemar test for the search predictions of the models [31]. First, a contingency table is created as:

	YAMNet Correct	Yamnet Incorrect
VAE Correct	a	b
VAE Incorrect	c	d

Table 2.2: Contingency table setup in the comparison of the two models.

From this table, the χ^2 statistic can be computed. Here, the continuity corrected version of the test statistic is used [32]:

$$\chi^2 = \frac{(|b-c|-1)^2}{(b+c)} \tag{2.11}$$

along with the p-value

$$p - \text{value} = 2\sum_{i=b}^{n} \binom{n}{i} 0.5^i (1-0.5)^{n-i} \tag{2.12}$$

The test is carried out at a significance level of $\alpha = 0.05$ and will conclude research question 2 regarding which learning method will produce the best vector representations for search and retrieval.

2.3 Audio Vector Representations

The general idea in this project is based on representing audio features in a lower dimensional space, only capturing the essential parts of the data. What these essential parts are can be hard to comprehend in a complex signal such as audio, which is why large learning models are used in this project. When considering a less complex example, a vector representation can for example describe a picture of an animal. The most important features to determine what animal was represented by the vector could include the number of legs, whether it has fur or not, and so on. The same concepts apply to the audio data, but these numerical features are not easy for

humans to perceive on their own. However, we can either perceive them when put in relation to other feature vectors (for example when clustering in a latent space) or when decoding and reconstructing the representations (as in the VAE model) [33].

One way of compressing audio data is to represent the most important features of the data in a latent space, i.e. by reducing the dimensionality of the data and only keeping the most vital and descriptive part of the signal. This can be done in a rudimentary simple manner, such as for example taking the mean of an audio signal and using that as a latent representation. This could also be done by performing Principal Component Analysis (PCA) to reduce dimensionality in the data. However, since these would be described as linear dimensionality reduction techniques, they have their limitations when modeling more complex and non-linear structures in the data.

The suggested approach in this project is therefore to use a DNN as a dimensionality reduction technique. A DNN structure is capable of capturing the complex and non-linear relationships in the often very high-dimensional data. A DNN can be trained in both a supervised and unsupervised manner and this project seeks to implement both learning methods for the representation of the audio data and compare them wrt. IR measures. For the approaches suggested here, the audio data is transformed as described in section 2.1 and is then passed as input to the DNN.

2.4 YAMNet

The YAMNet model [4] is developed by Google Research and is a deep learning model based on the MobileNet convolutional neural network architecture [34]. The reasoning

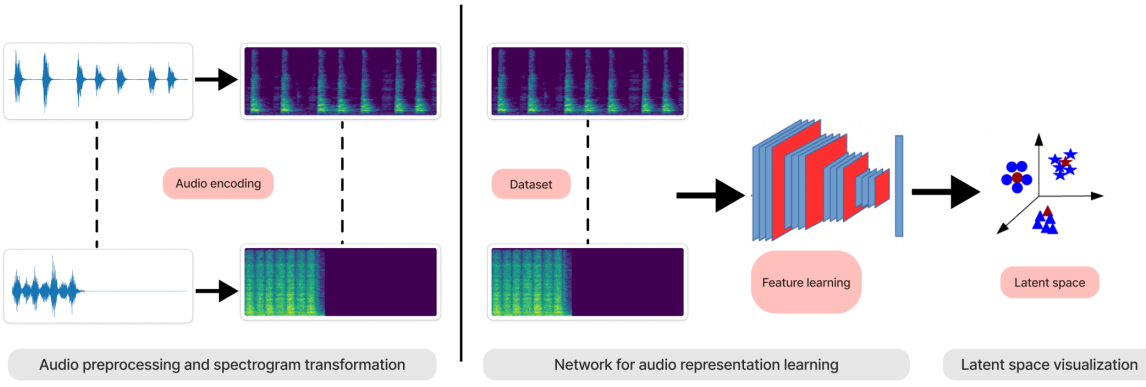

Figure 2.3: Overview of the methods used for compressing audio data to a latent space in the YAMNet model.

behind using YAMNet for generating vector representations includes its architecture with an embedding layer along with it being accessible and lightweight. Furthermore, it has provided fair results on benchmark tasks both on the AudioSet dataset [4] as well as other environmental datasets [35]. YAMNet is therefore used as a transfer learning model, which is trained on AudioSet and not on ESC-50. This may lead to an unfair advantage over the VAE, however it is still of interest to compare the models and asses whether it is possible to represent the audio in a meaningful way for similarity-based searching.

The architecture is based on depthwise separable convolutional layers, that have proven to obtain remarkable results, while still being smaller and faster than traditional convolutional neural networks. A total of 2 ”regular” convolutional layers and 13 depthwise convolution layers make up the model, which benchmarked around the same accuracy as VGG-16 and AlexNet while having only a fraction of the parameters. This drastic reduction in trainable parameters in the model is mainly due to the clever implementation of the depthwise separable convolutional layers compared to more parameter-heavy standard convolutional layers. By expressing the convolutions in this manner, a total reduction in computation is achieved with a factor of

$$\frac{1}{N} + \frac{1}{D_K^2} \tag{2.13}$$

where N is the number of output channels and D_K is the kernel size. The depthwise separable convolutional layer technique is visualized in 2.4. The model is designed for various audio classification tasks and it is trained discriminatively on 521 classes from

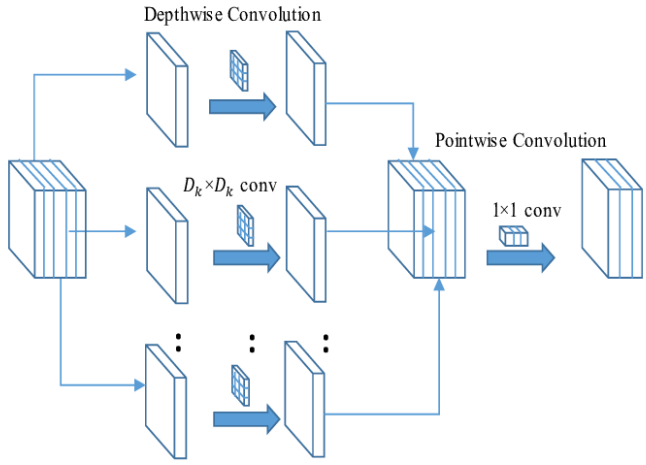

Figure 2.4: Depthwise seperable convolutional layers in the YAMNet model architecture. Figure from [36].

the AudioSet dataset [15], which consists of environmental recordings from YouTube videos. The YAMNet model takes the raw audio signal as input and generates a log Mel spectrogram as a part of the data preprocessing steps. These spectrograms are then processed using convolutional and pooling layers, that aid the model in recognizing and learning features in the audio signals.

YAMNet also utilizes an embedding layer, which is trained on the AudioSet data. The layers up to the embedding layer are implemented through transfer learning and maps the input in the model to a latent feature vector, which contains the most important features from the audio signal. This layer is the second-to-last layer and consists of 1024 features, meaning that the model compresses a spectrogram input into a 1024-dimensional vector.

For inputs with N frames per spectrogram, YAMNet outputs a $N * 1024$-dimensional vector. The architecture of the network is described in figure 2.5: In order to perform inference on the input and carry out a classification task, the vector representation is passed through a dense layer, which then outputs a probability distribution of predefined classes. The model proved to be very effective in audio classification tasks [38], which leads to the understanding that the latent vector from the embedding layer actually captured the features in the input well.

2.4.1 Selecting YAMNet features in latent space

Since the embedding layer captures the most important features, it is expected that similar features will have similar latent vectors. This means that input data that share the same semantic class (for example "Dog" for subclasses and "Animals" for superclasses) will most likely have embedded vectors that are close in the latent space. This observation is an essential part of the evaluation of the embeddings: if the latent

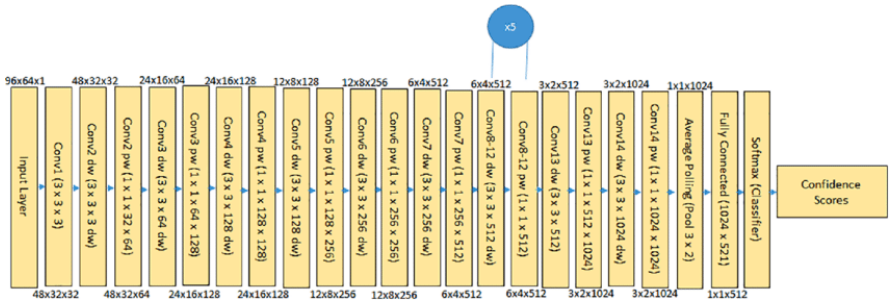

Figure 2.5: YAMNet Body Architecture. Note that the layers denoted `'dw'` are depthwise separable convolutional layers as depicted in figure 2.4. Figure from [37].

representations that are close to each other in the latent space share the same sub- or superclass, then the "important" features have been properly learned and the model fits well to the data. However, as audio data can vary both in length and size (recordings at different sample rates), the output of YAMNet will not always have the same dimensionality. Therefore, it is necessary to be able to compare an audio recording that only contains 2 frames and another that contains 15 frames with respectively a shape of (2×1024) and (15×1024). To ensure an equal comparison, the following is proposed:

By using measures that are representative of all frames and then concatenating them to a new representation, it is possible to compare representations of different dimensionalities.

1. Mean of features:
$$\boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{2.14}$$

2. Standard deviation of features:
$$\boldsymbol{\sigma} = \sqrt{\frac{\sum_{i=1}^{N} x_i - \mu}{N-1}} \tag{2.15}$$

3. Delta features: The delta features are computed using the Savitzky-Golay filter [39]:
$$\boldsymbol{\Delta}^{(m)} = \sum_{j=0}^{n} \left(\sum_{i=-\frac{m-1}{2}}^{\frac{m-1}{2}} C_i y_{j+i} \right), \qquad \frac{m-1}{2} \leq j \leq n - \frac{m-1}{2} \tag{2.16}$$

   where $\boldsymbol{\Delta}^{(m)}$ represents the m-th order derivative of the input data, y is the original input data, C_i are the filter coefficients, and m is the window size of the filter. The filter coefficients C_i are computed by solving a least-squares problem to obtain a polynomial that is the best approximation for the data points in the window of width m. These coefficients are then used to weigh neighboring samples in the window, essentially producing a smoothed estimate of the first derivative.

2.4.2 Principal Component Analysis

By passing an appropriately sized input ($N * M \gg 1024$ for an $N \times M$-sized Log Mel spectrogram input) through the YAMNet architecture, its dimensionality is reduced remarkably. However, it may be of interest to reduce it even more to assess vector representation clustering in the latent space. A way to do this is by applying

PCA to the processed latent representation and hereby gaining information regarding whether more similar audio inputs (audio input with the same sub- or superclass) are represented close in the latent space. If the classes are separated in some low dimensionality, then it is assumed that they are even better separated in higher dimensions via Cover's Theorem [40, 41]. Performing PCA on the latent representations from YAMNet can therefore be a good indicator of class separation. However, it is important to note that since the data is very high-dimensional, poorly separated clusters in a low dimensional space do not necessarily equal a bad separation in higher-dimensional space.

2.5 Variational Autoencoders

An Autoencoder (AE) is a neural network architecture, which is designed to learn in an unsupervised fashion. The AE encodes the input data to a latent space and then decodes it to a reconstruction. This reconstruction is then used to assess how well the model was able to represent the features in the latent space [42].

The architecture can be divided into two parts with the latent bottleneck in between. First, the input is passed to an *encoder* network, which translates the high-dimensional input into a lower-dimensional latent representation. This network can be described as an encoder function $g()$, parametrized by ϕ. Then, this representation can be regarded as a bottleneck in the network, $\mathbf{z} = g_\phi(\mathbf{x})$. This latent variable then feeds into the second part, the *decoder* network, described by a decoder function $f()$, parametrized by θ. Here, the decoder tries to reconstruct the original input based on the representation, i.e. the reconstructed input is $\hat{\mathbf{x}} = f_\theta(g_\phi(\mathbf{x}))$.

During the training of an autoencoder network, the parameters of the encoder and decoder functions, (ϕ, θ) are learned in order to recreate the input such that $\mathbf{x} \approx f_\theta(g_\phi(\mathbf{x}))$

The difference between the original input and the reconstruction can then be used as a metric to optimize the objective function in the network and can for example be a simple MSE loss:

$$\mathcal{L}_{AE}(\phi, \theta) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2 \tag{2.17}$$

The traditional autoencoder method can be extended to a variational Bayesian approach, where the input is mapped to a distribution instead of directly to a fixed latent vector [2, 3]. This also extends the name of the method to the Variational Autoencoder (VAE). For this purpose, the latent variable $\mathbf{z}$ is introduced along with a sample $\mathbf{x}$ from the conditional probability distribution $p_\theta(\mathbf{x}|\mathbf{z})$, which is parameterized by θ. The distribution is used to describe the relationship between input data $\mathbf{x}$ and the latent representation $\mathbf{z}$ using a prior, $p_\theta(\mathbf{z})$, a likelihood, $p_\theta(\mathbf{x}|\mathbf{z})$ and a

posterior probability distribution, $p_\theta(\mathbf{z}|\mathbf{x})$.

This probabilistic modeling allows for some control over how the latent distribution is modeled, for example, how many samples belonging to the same class are placed close together in the latent space. This combats the issue in non-variational AE, where the latent space is not easy to sample from or where it may not be continuous, essentially making the VAE preferable for this project.

In the VAE, we are interested in the posterior distribution, as this explains how likely the latent variable $\mathbf{z}$ is given the input $\mathbf{x}$. Since we can't calculate the posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ an approximative distribution, $q_\phi(\mathbf{z}|\mathbf{x})$, is introduced in order to approximate the posterior and we optimize ϕ so that $q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$. The two parameterizations of the distributions, $\{\theta, \phi\}$ are learned jointly during training to optimize an objective function [2]. To quantify how well $q_\phi(\mathbf{z}|\mathbf{x})$ approximates $p_\theta(\mathbf{z}|\mathbf{x})$ the Kullback-Leibler Divergence (D_{KL}) is used,

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) \tag{2.18}$$

Another way of approaching the VAE is by considering the parametric distribution p_θ. Assuming that the parameter that maximizes the likelihood for this distribution is θ^*, the model would create samples as such:

$$\mathbf{z}^{(i)} \sim p_{\theta^*}(\mathbf{z}) \tag{2.19}$$

$$\mathbf{x}^{(i)} \sim p_{\theta^*}(\mathbf{x}|\mathbf{z} = \mathbf{z}^{(i)}) \tag{2.20}$$

θ^* is the optimal parameter that satisfies the following:

$$\theta^* = \underset{\theta}{argmax} \prod_{i=1}^{n} p_\theta(\mathbf{x}^{(i)}) \tag{2.21}$$

$$= \underset{\theta}{argmax} \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)}) \tag{2.22}$$

In practice, we can only gain access to $\mathbf{x}$ in training as the sample from the latent variable so by marginalizing the likelihood wrt. $\mathbf{z}$, we get the marginal likelihood function in eq. 2.23. Calculating a value of θ directly would be possible as long as it is tractable to compute the marginal likelihood:

$$p_\theta(\mathbf{x}^{(i)}) = \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z} = \mathbb{E}_{\mathbf{z}\sim p_\theta(\mathbf{z})}[p_\theta(\mathbf{x}^{(i)}|\mathbf{z})] \tag{2.23}$$

Unfortunately, the calculation of the complete integral is often intractable and another way of estimating the optimal parameter. The D_{KL} in eq. 2.18 and the expectation value in 2.23 are the key components in the Evidence Lower Bound (ELBO), which is defined in equation 2.24 and will be derived in section 2.5.2.

While introducing these parametrical probability distributions, is important to keep track of what these parametrizations actually denote. In the prior, $p_\theta(\mathbf{z})$, the notation θ adheres to standard distribution notation $p_\theta(\mathbf{z}) = \mathcal{N}(\mu, \sigma)$ where $\theta = \{\mu = 0, \sigma = 1\}$. However, the rest of the notations $\{\theta, \phi\}$ will refer to the set of parameters of respectively the stochastic encoder network and the stochastic decoder network that are learned jointly during training. This notation can be shown as the following for the decoder D_θ and the encoder E_ϕ:

where $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu} = D_\theta(\mathbf{z}), \boldsymbol{\sigma} = D_\theta(\mathbf{z}))$ and $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu} = E_\phi(\mathbf{x}), \boldsymbol{\sigma} = E_\phi(\mathbf{x}))$.

The overall structure of a VAE model can be seen in figure 2.7, where it is evident that the VAE consists of two sections, much like the AE. First, a stochastic encoder $q_\phi(\mathbf{z}|\mathbf{x})$ is used for mapping the input to the latent space. This network is similar to what the encoder function g_ϕ does in the non-variational AE, however since it is probabilistic, it instead uses the approximative posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$. The second part of the model is a stochastic decoder $p_\theta(\mathbf{x}|\mathbf{z})$, that decodes the latent sample to a reconstruction used in the objective function. This can be seen as similar to the decoder function f_θ in the AE, however still using a probability distribution. The decoder part of the network is visualized in figure 2.6.

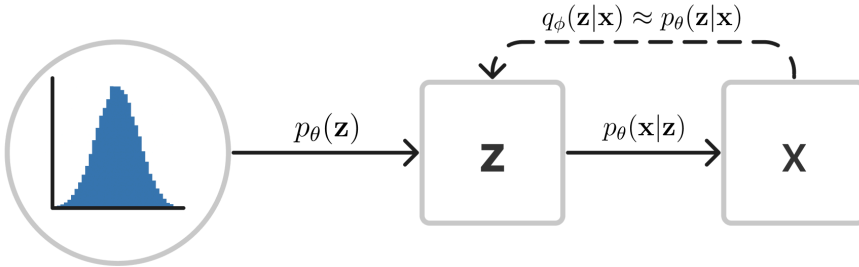

Figure 2.6: The generative model involved in the VAE. The solid black arrows denote usage of the distribution $p_\theta(\cdot)$ and the dashed black arrow denotes usage of the approximative distribution $q_\phi(\cdot)$ that approximates the intractable posterior $p_\theta(\mathbf{z}|\mathbf{x})$.

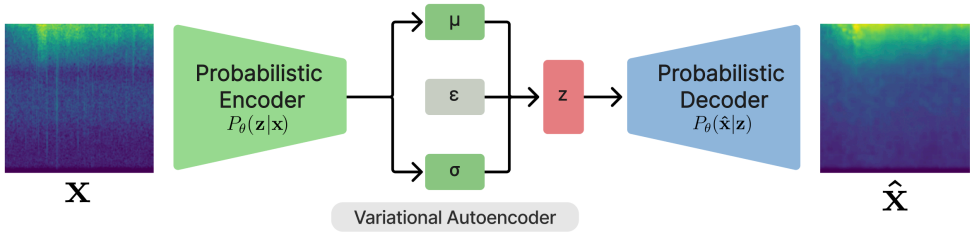

Figure 2.7: The Variational Autoencoder structure on a high level. Note that $\mathbf{x}$ and $\hat{\mathbf{x}}$ are not identical, since $\hat{\mathbf{x}}$ is the reconstruction of $\mathbf{x}$ after the network is trained. These two spectrograms should ideally be identical and they are the ones used in the reconstruction loss in section 2.5.1.

2.5.1 ELBO Objective Function

The following section will first outline the objective function of the VAE when training. The reader will first be presented with the entire derived objective function and will then be carried through the derivation of this. Finally, the actual implementation of the objective function is presented.

The objective function of a VAE is made up of two main parts - a reconstruction part that serves as to penalize bad reconstructions in the output and a regularization part that aims to enforce a preset prior distribution to the latent space, here a multivariate Gaussian distribution. The first term serves as a reconstruction loss just like in AEs, but is instead based on the expected value of the logarithm of the conditional probability distribution $p_\theta(\mathbf{x}^{(i)}|\mathbf{z})$, i.e. $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)}[\log p_\theta(\mathbf{x}_i \mid \mathbf{z})]$. The second term comes from computing the D_{KL} between the estimated distribution from the decoder, $q_\phi(\mathbf{z}|\mathbf{x})$ and the true posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$. As D_{KL} is used to measure the loss of information whenever we use distribution Y to represent distribution X, this term will measure how well we represent the true posterior with the approximation. Optimizing this term will therefore be by minimizing it.

The ELBO is a lower bound on the marginal likelihood of the observed data in the model. When the audio signals are compressed into latent space, the vectors follow a certain distribution, which is based on the features learned by the deep neural network. Optimally, this latent vector would follow the true distribution of the audio signal in the latent space, since this would allow us to perfectly sample and recreate the data - which is the goal in the VAE. However, since the exact computation of the true distribution is intractable, we need to find a good approximation for this. The ELBO gives a lower bound of how good this approximation is and by maximizing the ELBO, we get a better lower bound for the variational inference.

The ELBO is defined as [2]:

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \parallel p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)}[\log p_\theta(\mathbf{x}_i \mid \mathbf{z})] \tag{2.24}$$

In the specific case where the prior over the latent variables is a multivariate Gaussian and thereby $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$, then $p_\theta(\mathbf{x}|\mathbf{z})$ is also set to be a multivariate Gaussian since the data is real-valued. It is also assumed that the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ follows an approximate multivariate Gaussian with an approximate diagonal covariance. This assumption is needed since $p_\theta(\mathbf{z}|\mathbf{x})$ is still unknown and we are unable to compute it.

2.5.2 Derivation of the ELBO

In the notation of $D_{KL}(P||Q)$, D_{KL} quantifies how well the distribution Q models the actual distribution P. Therefore, it plays an important role in the ELBO, since minimizing the D_{KL} here is equivalent to maximizing the log-likelihood that the approximative distribution models the true posterior of the data.

Using the reversed D_{KL}, $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x}))$ as a measurement between the two distributions q_ϕ, parameterised by ϕ, and p_θ, parameterised by θ:

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})) = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} \tag{2.25}$$

The conditional probability $p_\theta(\mathbf{z}|\mathbf{x})$ can be written as $p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{x})}$. Substituting into eq. 2.25 and distributing the terms [42]:

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}|\mathbf{x})) = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} d\mathbf{z} \tag{2.26}$$

$$= \int q_\phi(\mathbf{z}|\mathbf{x}) (\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} + \log p_\theta(\mathbf{x})) d\mathbf{z} \tag{2.27}$$

$$= \left(\int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} d\mathbf{z} \right) + \left(\int \log p_\theta(\mathbf{x}) q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} \right) \tag{2.28}$$

$$= \left(\int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} d\mathbf{z} \right) + \left(\log p_\theta(\mathbf{x}) \int q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} \right) \tag{2.29}$$

Since $\int q_\phi(\mathbf{z}|\mathbf{x})d\mathbf{z} = 1$, we can continue with:

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}|\mathbf{x})) = \log p_\theta(\mathbf{x}) + \left(\int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} \right) \tag{2.30}$$

$$= \log p_\theta(\mathbf{x}) + \left(\int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})} d\mathbf{z} \right) \tag{2.31}$$

$$= \log p_\theta(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - \log p_\theta(\mathbf{x}|\mathbf{z})] \tag{2.32}$$

$$= \log p_\theta(\mathbf{x}) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] \tag{2.33}$$

This concludes as

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}|\mathbf{x})) = \log p_\theta(\mathbf{x}) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] \tag{2.34}$$

$$\log p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) \tag{2.35}$$

Since we do not have access to the true posterior $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}|\mathbf{x}))$, we cannot compute the LHS of eq. 2.35. However, since this term is always ≥ 0, we can disregard this and instead represent the RHS as a lower bound on the log-likelihood:

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) \tag{2.36}$$

The LHS of equation 2.36 represents the term that we are interested in maximizing during training and when learning the true distributions. The objective of the training is to maximize the log-likelihood ($\log p_\theta(\mathbf{x})$) when generating data. Negating equation 2.36 therefore brings us back to the objective for the variational Bayes approach:

$$\mathcal{L}(\theta, \phi) \leq \log p_\theta(\mathbf{x}) \tag{2.37}$$

$$\mathcal{L}(\theta, \phi) = -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) \tag{2.38}$$

$$\theta^*, \phi^* = \underset{\theta,\phi}{\operatorname{argmin}}\, \mathcal{L}(\theta, \phi) \tag{2.39}$$

which is equivalent to the ELBO, described in eq. 2.24.

Kingma & Welling [2] provides a solution of the term $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}))$ in the case of the multivariate Gaussian as such:

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) = \frac{1}{2} \sum_{j=1}^{J} (1 + \log((\sigma_{btl,j})^2) - (\mu_{btl,j})^2 - (\sigma_{btl,j})^2) \tag{2.40}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are obtained in the variational model from the latent layer. The *btl* subscript in $\mu_{btl,j}$ and $\sigma btl, j$ denotes that these are the mean and variance values in the bottleneck layer of the VAE. This subscript is introduced to not confuse them with $\mu_{recon,l}$ and $\sigma_{recon,l}$ from the output layer (see eq. 2.49). The solution in equation 2.40 is the one used directly in the experimental setup of the VAE.

2.5.3 Squared Error Derivation

The decoder follows a multivariate Gaussian, with the covariance matrix $\mathbf{C} = \boldsymbol{\sigma}^2\mathbf{I}$ which is defined as

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{C}) \tag{2.41}$$

$$= \frac{1}{(2\pi)^{L/2}\sqrt{det(\mathbf{C})}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{2.42}$$

where

$$det(\mathbf{C}) = \prod \sigma_i^2 \tag{2.43}$$

By taking the logarithm on both sides, we get

$$\log p(\mathbf{x}|\mathbf{z}) = \log\left(\frac{1}{(\pi)^{L/2}\sqrt{det(\mathbf{C})}}\right) + \sum \frac{(x_i - \mu_i)^2}{-2\sigma_i^2} \tag{2.44}$$

$$= \frac{-L}{2}\log(2\pi) + \frac{-1}{2}\log(\prod \sigma_i^2) + \sum \frac{(x_i - \mu_i)^2}{-2\sigma_i^2} \tag{2.45}$$

$$= \frac{-L}{2}\log(2\pi) + \frac{-1}{2}\sum \sigma_i^2 + \sum \frac{(x_i - \mu_i)^2}{-2\sigma_i^2} \tag{2.46}$$

Since the dimension of the Gaussian, L is fixed in this setting as well as the variance, $\boldsymbol{\sigma}$, they can be disregarded when we seek a function to optimize. We therefore get

$$\log p(\mathbf{x}|\mathbf{z}) = \sum \frac{(x_i - \mu_i)^2}{-2\sigma_i^2} + k \tag{2.47}$$

k is the disregarded constants.

Maximizing the log-likelihood term, therefore, corresponds to minimizing $\sum \frac{(\mathbf{x}-\boldsymbol{\mu})^2}{2\boldsymbol{\sigma}^2}$, which is a squared error loss divided by the variance. This means, that by letting $p_\theta(\mathbf{x}|\mathbf{z})$ follow a multivariate Gaussian distribution, it is possible to approximate $\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$ by Monte Carlo Approximation since we can sample from the multivariate Gaussian distribution. This is done using a single sample in the bottleneck $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. Since the Monte Carlo approximation is based on a single sample, it is possible to implement the second loss term as a squared error divided by the variance scaled with a factor of 2 when training the VAE model.

2.5.4 The ELBO revisited

By using analytical solutions and simplifications of the terms in the ELBO (eq. 2.24), we arrive at something that looks like the variational lower bound by for the VAE

that Kigma et al [2] originally proposed:

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} \Big(1 + \log((\sigma_{btl,j}^{(i)})^2) - (\mu_{btl,j}^{(i)})^2 - (\sigma_{btl,j}^{(i)})^2\Big) + \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(\mathbf{x}^{(i)} | \mathbf{z}^{(i,l)}) \tag{2.48}$$

With the derivation of the mean squared error loss instead of the last term in eq. 2.48, the implemented ELBO for the experimental setup will be:

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} \Big(1 + \log((\sigma_{btl,j}^{(i)})^2) - (\mu_{btl,j}^{(i)})^2 - (\sigma_{btl,j}^{(i)})^2\Big) + \frac{1}{L} \sum_{l=1}^{L} \frac{(x_l^{(i)} - \mu_{recon,l}^{(i)})^2}{\sigma_{recon,l}^{(i)^2}} \tag{2.49}$$

$$\simeq \frac{1}{2} \sum_{j=1}^{J} \Big(1 + \log((\sigma_{btl,j}^{(i)})^2) - (\mu_{btl,j}^{(i)})^2 - (\sigma_{btl,j}^{(i)})^2\Big) + \frac{(x^{(i)} - \mu_{recon}^{(i)})^2}{\sigma_{recon}^{(i)^2}} \tag{2.50}$$

where $\mu_{btl,j}$ and $\sigma_{btl,j}^2$ denotes the j-th index of the bottleneck vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ which are learned during training and also parameterize the sampling step in the VAE. μ_{recon} and σ_{recon}^2 denotes the mean and variance values in the output layer, which is the reconstruction in the network. Note, that L is the number of samples in the Monte Carlo approximation of the expectation value. In practice, one sample is sufficient for this approximation and the expression $1/L \sum_{l=1}^{L}$, therefore, disappears in eq. 2.50.

2.5.5 The reparametrization trick

Consider the probabilistic objective function of the form [43]

$$\mathcal{F}_\theta := \int p_\theta(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} = \mathbb{E}_{p_\theta(\mathbf{x})}[f(\mathbf{x})] \tag{2.51}$$

In model training, i.e. learning the distributional parameters θ, the gradients of eq. 2.51 becomes of interest, since these are backpropagated through the model when updating model weights. The gradient of these distributional parameters is denoted

$$\boldsymbol{\eta} := \nabla_{\theta,\phi} \mathcal{F}_\theta = \nabla_{\theta,\phi} \mathbb{E}_{p_\theta(\mathbf{x})}[f(\mathbf{x})] \tag{2.52}$$

The loss function, f, is based on samples generated from the approximative function $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x})$. However, since this sampling is a stochastic process, it is impossible to backpropagate the gradient through the sampling, making it unsuitable for training and learning. However, the originally random variable $\mathbf{z}$ can be expressed as a deterministic variable $\mathbf{z} = g_\theta(\mathbf{x}, \boldsymbol{\epsilon})$, where the $\boldsymbol{\epsilon}$-term is introduced as an auxiliary variable from a unit Gaussian distribution. This means that g_θ samples from the distribution of ϵ to $\mathbf{x}$ while being parameterized by θ.

This allows for the conversion

$$\mathbb{E}_{p_\theta(\mathbf{x})}[f(\mathbf{x})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g_\theta(\boldsymbol{\epsilon}))] \tag{2.53}$$

Now, the expectation value is independent of the distributional parameters θ and we can therefore derive the gradient of eq 2.51 by reparameterizing:

$$\boldsymbol{\eta} = \nabla_\theta \mathbb{E}_{p_\theta(\mathbf{x})}[f(\mathbf{x})] = \nabla_\theta \int p_\theta(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} \tag{2.54}$$

$$= \nabla_\theta \int p_\theta(\boldsymbol{\epsilon}) f(g_\theta(\boldsymbol{\epsilon})) d\boldsymbol{\epsilon} \tag{2.55}$$

$$= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_\theta f(g_\theta(\boldsymbol{\epsilon}))] \tag{2.56}$$

From this, the estimator $\boldsymbol{\eta}$ can then be estimated using Monte Carlo methods. This estimation then allows for backpropagation, meaning that the model can update its weights when reparameterizing the probabilistic system.

In the case of variational inference, we can split this expectation into two terms:

$$\nabla_\theta \mathcal{L}_{\theta,\phi} = \nabla_\theta \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \tag{2.57}$$

$$\nabla_\phi \mathcal{L}_{\theta,\phi} = \nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \tag{2.58}$$

The Leibniz integral rule allows for the gradient to be pushed into the expectation since it is not parameterized by θ:

$$\nabla_\theta \mathcal{L}_{\theta,\phi} = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\nabla_\theta(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))] \tag{2.59}$$

$$= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\nabla_\theta(\log p_\theta(\mathbf{x},\mathbf{z})] \tag{2.60}$$

where the term not parameterized by θ can be disregarded when considering the gradient. Then, we can take the integral over the expectation, as $q_\phi(\mathbf{z}|\mathbf{x})$ is continuous:

$$\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\nabla_\theta(\log p_\theta(\mathbf{x},\mathbf{z})] = \nabla_\theta \int q_\phi(\mathbf{z}|\mathbf{x})(\log p_\theta(\mathbf{x},\mathbf{z}) \tag{2.61}$$

$$= \int q_\phi(\mathbf{z}|\mathbf{x}) \nabla_\theta(\log p_\theta(\mathbf{x},\mathbf{z})) \tag{2.62}$$

which is a tractable integral. This gradient push-in however, is not allowed when considering the second expectation term:

$$\nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})]$$

since the gradient now is wrt. ϕ that the expectation also depends on. However, by introducing the reparametrization, the expectation is no longer dependent on ϕ and

the gradient can be pushed in to bring us a tractable integral:

$$\nabla_\phi \mathcal{L}_{\theta,\phi} = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\nabla_\phi(\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}))] \tag{2.63}$$
$$= \mathbb{E}_{\epsilon\sim p(\epsilon)}[\nabla_\phi(\log q_\phi(\mathbf{z}|\mathbf{x}))] \tag{2.64}$$
$$= \nabla_\phi \int p(\epsilon)(\log q_\phi(\mathbf{z}|\mathbf{x})) \tag{2.65}$$
$$= \int p(\epsilon)\nabla_\phi(\log q_\phi(\mathbf{z}|\mathbf{x})) \tag{2.66}$$

where $\mathbf{z}$ is expressed in terms of the parameters of the multivariate Gaussian distribution:

$$\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I}) \tag{2.67}$$
$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{2.68}$$

Here, $\odot$ is the element-wise product of $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim p(\epsilon) = \mathcal{N}(0, \mathbf{I})$. The reparametrization trick is visualized in figure 2.8:

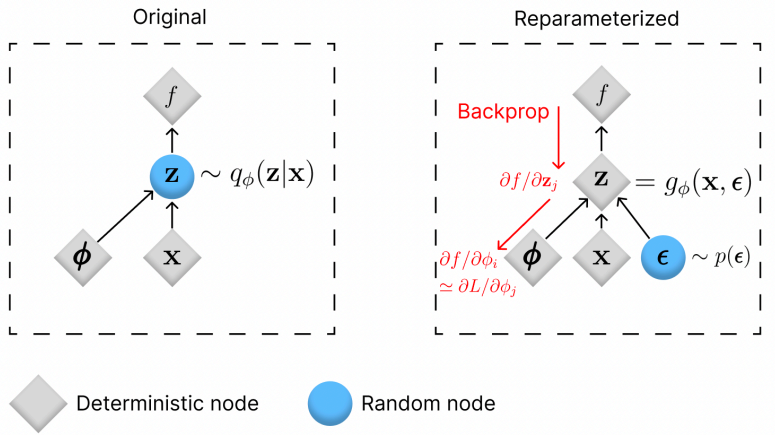

Figure 2.8: The reparametrization trick: By introducing a new parameter $\boldsymbol{\epsilon}$, it is possible to reparameterize $\mathbf{z}$ in a way that allows backpropagation to flow through the deterministic nodes to update the gradients properly. The figure is based on the reparameterization introduced by Kingma and Welling (2013) [2] and Rezende (2014) [3].

2.5.6 Gated Linear Unit Activation Function

The Gated Linear Unit (GLU) activation function [44] is introduced as an attempt to capture dependencies in time over longer ranges. The GLU function allows the VAE to selectively filter relevant features from the data through its gating mechanism.

Inspired by the gating in LSTM-networks, the GLU function can either allow or block information flow from layer to layer, making it an advantageous activation function for VAEs compared to the Rectified Linear Unit (ReLU) activation function typically used in DNN models. The GLU function is defined as

$$h_l(\mathbf{X}) = (\mathbf{X} * \boldsymbol{W} + \mathbf{b}) \otimes \sigma(\mathbf{X} * \boldsymbol{V} + \mathbf{c}) \tag{2.69}$$

where σ is the sigmoid function, $\sigma(x) = \dfrac{e^x}{e^x + 1}$, $\mathbf{W}$ and $\mathbf{V}$ are sets of weights and $\mathbf{b}$ and $\mathbf{c}$ are their respective biases.

The GLU function is used in each layer block in the architecture of the VAE. This architecture is shown in figure 2.9.

2.5.7 Batch Normalization

When training a DNN model, the inputs to each layer are provided by the previous one. However, since these inputs change during training it might be difficult for the network to converge and thereby slowing the training process down. This phenomenon is known as the ”Internal Covariate Shift” and describes the change in the distribution of the inputs to each layer. As a solution to combat this, batch normalization was introduced to normalize the inputs given to each layer by adjusting the mean and variance of these [45]. This normalization layer becomes a part of the model architecture and can also act as a regularizer to combat the potential overfitting of the model by stabilizing the training process.

In the specific setting of this project, each layer block finishes with a batch normalization layer to ensure stability in the inputs to the next layer block. This can also be seen in the architecture diagram in figure 2.9.

The architecture diagram of the VAE model concludes the theoretical foundation behind the two models investigated in this project along with how they are going to be evaluated and compared. To score them and evaluate them, we first have to create an experimental setup for them. The next chapter will describe exactly how the models are configured and trained and will also include descriptions of how the vector representations are generated and used for search and retrieval.

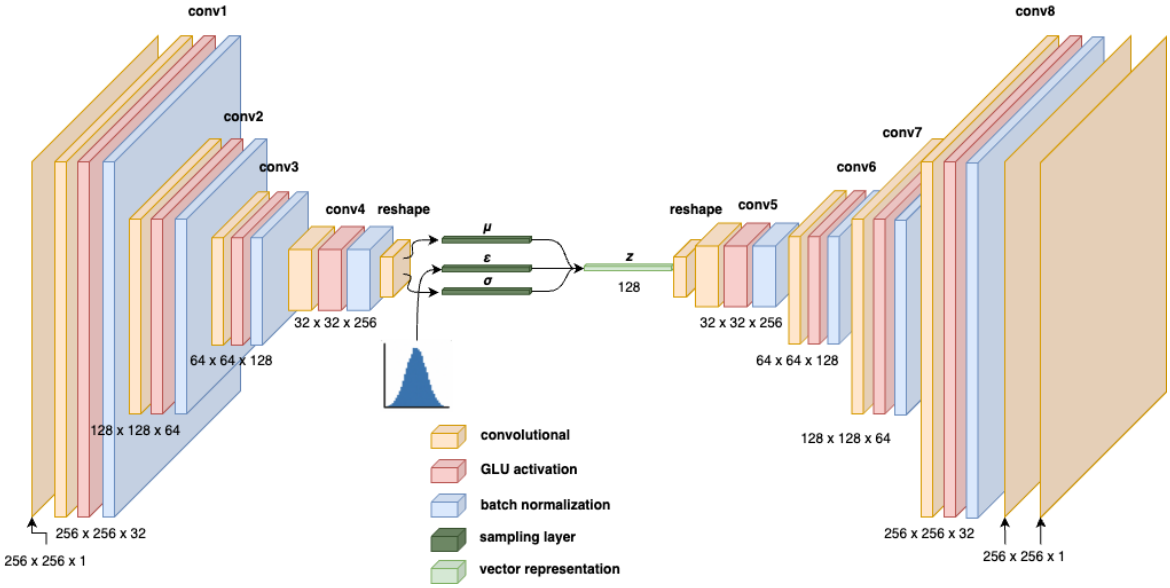

Figure 2.9: Architecture diagram for the VAE used in this project.

CHAPTER 3

Experiments

This chapter describes the experiments that are carried out in order to answer the two research questions in section 1.2.

First of all, the experimental configurations of both models are described as well as how the vector representations of the audio are generated including feature selection of these. The representations are then used for a search and retrieval experiment. The outcome of this experiment will assess how well these representations will perform in terms of IR to retrieve relevant audio recordings as it was stated in research question 1.

This chapter will also describe how the comparison of the two learning methods will be carried out in order to answer research question 2.

3.1 Supervised Learning Approach

The supervised learning approach in this project is based on the YAMNet model as described earlier. The following describes how the vector representations are generated and how the features in these representations are going to be selected. Then, the experimental setup for the similarity-based search is described along with how this is going to be evaluated. Since the Google Research team hasn't released YAMNet for training but only for inference [46], it is here used as a transfer learning model with frozen weights for embedding generation.

It is very important to note in the following that since YAMNet is here used as a transfer learning model, it has been trained beforehand by the Google Research team on the large AudioSet dataset and not on the ESC-50 dataset, however, it is still being evaluated on the vector representations it generates from the latter, which is previously unseen for the model.

3.1.1 Architecture

This project uses the YAMNet model as it was in the original implementation by Google [1]. The data is preprocessed as described in section 2.1.1, where each wave-

form is standardized and converted to mono and is then passed through the network pipeline that includes the transformation into a log Mel spectrogram. A vector representation of each input is then obtained from the second-to-last layer. When using the model as a transformer model with pre-trained weights, the frozen weights and model parameters are loaded into the architecture. Initializing the model using these weights, the input flows through the model, where the vector representations are generated by pooling with a global average on the third-to-last layer and saved to local storage for analysis.

3.1.2 Representing vector embeddings

The vector embeddings are further reduced in dimensionality since the output dimensionality of YAMNet may vary depending on the input size, as described in section 2.4.1. To assess which combinations of the 3 representations yield the most informative output for searching, an experiment is set up to compare the accuracies of these different methods. The combination of representations that yield the best accuracy, which is used since the classes are perfectly balanced, will then be selected to represent the embeddings when performing search and retrieval.

3.1.3 Searching

After selecting features, each embedding vector is then compared to the rest in a one-vs-all search. Searching in the embedded vector space is a way of assessing the quality of the class separation in the latent space in a quantitative way. The pseudo-code in algorithm 1 outlines a search from a collection of embeddings.

3.1.4 PCA

As the dimensionality of the latent space output from the YAMNet model is of size $N \times 1024$ it is challenging to visualize whether the classes have been separated properly. By carrying out the PCA as mentioned in section 2.4.2, it becomes possible to perform this visualization. However, even though PCA is a technique for reducing dimensionality, it still struggles with very high-dimensional data [47]. For the case with a $N \times 1024$ representation, this may be the case if the goal is to reduce it to fewer dimensions for the assessment of class separation. The issue with high-dimensional data in a PCA is that a lot of the data often has a lot of features with very low variance. Especially in this case, where the inputs are min-max normalized to a range of $[0, 1]$, it may be the case that some of the variances in a PCA will be redundant. The variance of the embedding features is distributed as shown in figure 3.1. The distribution of these variances serves as a motivation for a thresholding procedure, whereby low-variance features in the embeddings are removed from the data before the PCA. Since the principal components in a PCA are selected based on their contribution to the total variance, having higher variance features would most likely serve

Algorithm 1 Searching Latent Space (YAMNet embeddings)

Require: Frozen YAMNet weights

Pass input through YAMNet model and collect embeddings $\in \mathrm{R}^{N \times M}$ in a list from second-to-last layer

$y, \hat{y}_{euc}, \hat{y}_{cos}$ = Empty arrays

for embedded vector in a list of embeddings **do**

Append sub/super-label to y

Get vector $\mathbf{x}_{euc}$ which satisfies $\underset{\mathbf{x}}{\operatorname{argmin}} \sqrt{\sum_{i=1}^{n} (q_i x_i)^2}$ ▷ Euclidian Distance

Get vector $\mathbf{x}_{cos}$ which satisfies $\underset{\mathbf{x}}{\operatorname{argmin}} \frac{\sum_{i=1}^{n} q_i x_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \sqrt{\sum_{i=1}^{n} x_i^2}}$ ▷ Cosine Similarity

Use the most similar vectors to assign predicted labels:

Append label of embedded vector $\mathbf{x}_{euc}$ to $\hat{y}_{euc}$

Append label of embedded vector $\mathbf{x}_{cos}$ to $\hat{y}_{cos}$

end for

Score the search using appropriate IR metrics using $y, \hat{y}_{euc}$ and $\hat{y}_{cos}$

as a better starting point for the analysis. The thresholding procedure removes all features in the embedding that contribute to a variance lower than the threshold and creates a representation with fewer features to use in the PCA. The thresholding is performed using the feature selection function `VarianceThreshold` from the `sklearn` library [48]. The thresholding values range from 0.0000 to 0.0025 with a step size of 0.0001, resulting in 25 new representations. These thresholded representations are then classified using a multinomial logistic regression and a Support Vector Machine (SVM) classifier with a linear kernel to assess whether removing features with low variance impacts the quality of the PCA and thereby separates the classes more in the latent space.

3.1.5 Missing YAMNet labels

Since the YAMNet model is trained discriminatively on the AudioSet dataset, it has seen a lot of classes in pre-training that are not present in the ESC-50 dataset. It is also presented with new classes present in ESC-50 that are absent in AudioSet during fine-tuning. This could potentially mean that YAMNet would have worse performance on the previously unseen classes compared to classes that are present in both datasets. Therefore, it is of interest to compare the classification of embeddings belonging to these classes.

The classes in ESC-50 that do not have an equivalent class in AudioSet are shown in table 3.1 along with their semantic superclass.

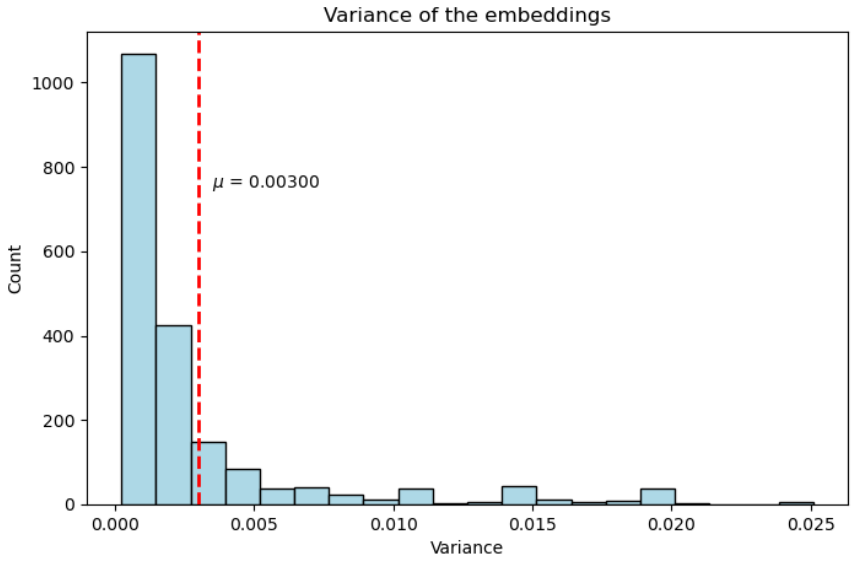

Figure 3.1: Distribution of the variance of the **Δ** features embeddings generated by YAMNet.

Animals	Natural soundscapes & water sounds	Human, non-speech sounds	Interior/domestic sounds	Exterior/urban noises
Cow	-	Drinking/sipping	Can opening	-
Hen	-	-	Washing machine	-

Table 3.1: Semantic classes that are present in the ESC-50 dataset without an equivalent semantic class in the AudioSet dataset.

3.2 Unsupervised Learning Approach

The unsupervised learning approach is based on the VAE model as described earlier. The goal of the unsupervised learning approach is, just as it is with the supervised learning approach, to assess whether it is possible to generate vector representations of the spectrograms and use these for search and retrieval.

3.2.1 VAE

The VAE in this project consists of an input layer, 4 grouped convolutional layers in the encoder, a dense layer in the bottleneck, a sampling layer, 4 grouped transpose convolutional layers in the decoder, and finally a transpose convolutional layer

for output in the decoder. These layers are shown in figure 2.9, where the grouped convolutional layers consist of a convolutional layer with kernel size 3×3, a GLU activation function, and a batch normalization layer.

Implementing a custom training and testing step in Keras, the model is trained on 1000 epochs with a standard K-fold Cross Validation (CV) with K=5 to ensure that the model fits equally well on all parts of the data. The VAE model is implemented with the following hyperparameters:

- "latent_space_dim": Dimensionality of the latent bottleneck in the bottleneck. This dimensionality also automatically decides the dimensionality of the $\boldsymbol{\mu}_z, \boldsymbol{\sigma_z}$ and z sampled from the sampling layer. The dimensionality hyperparameter will be tuned over several runs. The dimensionalities investigated are $\{32, 64, 128, 256\}$.
- "learning_rate": Learning rate that the optimizer learns with. The optimizer used is Adaptive Moment Estimation (Adam) [49]. For this project, the learning rate is set to 0.0005 as this value yielded better optimization traces in initial runs when compared to 0.001 and 0.0001.
- "batch_size": The batch size for training. Research suggests that smaller batches can improve accuracies, especially when dealing with single images (spectrograms) as input [50]. The value of this hyperparameter is also based on computational resources available as well as time. The batch size is set to 20 in this project, as this gave reasonable training times for the model.
- "epochs": The number of epochs trained in each CV fold. In the runs described in this project, the model was trained for 1000 epochs for each CV fold.
- "run_name": Name of the run used for logging and saving purposes
- "beta_warmup": Boolean parameter that decides whether or not the training should utilize a β-warmup strategy for the first 10 epochs [51]. This strategy is described further below.

After training the model, a visual reconstruction of 5 inputs from a previously unseen validation part of the full dataset is computed in order to visually evaluate the quality of the reconstructions of the model.

The vector representations generated from passing an input spectrogram through the encoder are also used for searching, just as in the supervised learning method. However, here they are generated in the bottleneck layer of the model. This bottleneck layer outputs 2 things, namely the mean and the variance representations of the model input. The mean values, denoted $\boldsymbol{\mu}_z$, are used as the vector representations since the variance term $\boldsymbol{\sigma}_z^2$ is essentially just noise which adds to the robustness

of the multivariate Gaussian that these two values parameterize. This procedure is described in the pseudocode in algorithm 2.

Algorithm 2 Searching Latent Space (VAE embeddings)

Require: Saved VAE model weights, saved VAE model parameters

Pass input through encoder part of VAE to generate a vector representation of $\boldsymbol{\mu}_z \in \mathbb{R}^N, N =$"latent_space_dim". Use $\boldsymbol{\mu}_z$ as the vector representation and collect the embeddings in a list

$y, \hat{y}_{euc}, \hat{y}_{cos} =$ Empty arrays

for embedded vector in a list of embeddings **do**

Append sub/super-label to y

Get vector $\mathbf{x}_{euc}$ which satisfies $\underset{\mathbf{x}}{\operatorname{argmin}} \sqrt{\sum_{i=1}^{n} (q_i x_i)^2}$ ▷ Euclidian Distance

Get vector $\mathbf{x}_{cos}$ which satisfies $\underset{\mathbf{x}}{\operatorname{argmin}} \frac{\sum_{i=1}^{n} q_i x_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \sqrt{\sum_{i=1}^{n} x_i^2}}$ ▷ Cosine Similarity

Use the most similar vectors to assign predicted labels:

Append label of embedded vector $\mathbf{x}_{euc}$ to $\hat{y}_{euc}$

Append label of embedded vector $\mathbf{x}_{cos}$ to $\hat{y}_{cos}$

end for

Score the search using appropriate IR metrics using $y, \hat{y}_{euc}$ and $\hat{y}_{cos}$

Fit a linear classifier to the latent vectors in order to assess label separation in higher dimensions

3.2.1.1 ELBO in practice

Training of a VAE model seeks to minimize or tighten the lower bound on the marginal likelihood as mentioned in section 2.5.1. This means that the learned parameters during training are used as an estimate of this bound, measured during training.

From the derivations made in section 2.5.2, under the assumption that the output follows a multivariate Gaussian distributed [2], we arrive at eq. 2.50:

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma_{btl,j}^{(i)})^2) - (\mu_{btl,j}^{(i)})^2 - (\sigma_{btl,j}^{(i)})^2\right) + \frac{(x^{(i)} - \mu_{recon}^{(i)})^2}{\sigma_{recon}^{(i)2}} \tag{3.1}$$

All of these parameters are learned during training where some of them, $\mu_{btl}, \sigma_{btl}^2 \in \mathbb{R}^N$ where N is the dimensionality of the bottleneck, are learned in the layer right before the sampling layer. $\mu_{recon}, \sigma_{recon}^2 \in \mathbb{R}^{(256,256,1)}$ are learned in the two output layers.

Note, that there are no issues with the dimensionality differences across the two terms since they are calculated as two terms and added thereafter.

3.2.1.2 β-warmup

As an attempt to prevent exploding values of D_{KL} in training, Sønderby et al proposed a linear warmup sequence for the first N epochs of a model training run [51]. This method scales the regularizing D_{KL} term in the ELBO linearly with the current epoch number, k, and gently introduces it into the objective function:

$$\beta = \frac{1}{N} \cdot k, \quad k \in \{1, 2, ..., N\} \tag{3.2}$$

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = -\beta D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \parallel p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)}[\log p_\theta(\mathbf{x}_i \mid \mathbf{z})] \tag{3.3}$$

The setup in this project uses $N = 10$ epochs for the warmup sequence in runs where enabled.

3.2.1.3 Linear classification in high-dimensional space

Since the clustering in the latent space is essential for the search to be accurate, it is important to assess how well the vectors are separated in that space. This is done by fitting linear models to the (relatively) high-dimensional latent space and observing the accuracy of this. The chosen linear models are multinomial logistic regression, suitable for multi-class problems as well as a SVM classifier with a linear kernel. The classifiers are fitted over 5 folds to yield a mean accuracy as the result.

As the latent layer outputs two different things, the mean and the standard deviation of the latent representation, it is important to keep track of where the information from the recording is represented the best. In the following, the mean values in the bottleneck are used as the vector representations for the VAE model.

3.3 Model assessment

The models will be assessed based on their IR scores, which will be the ones described in 2.2.1. In IR, it is uncommon to use labels as the scores will be based on how relevant the retrieved items are when querying. However, this relevance is hard to determine unsupervised as it would require either a human or another model to decide this. Therefore, the semantic classes from section 2.1 are used as labels for each audio recording. This also allows for a divide into two sets of super- and subclasses, with respectively 5 and 50 total classes in each set. The similarity-based searching experiment is then essentially a classification using the KNN algorithm, where $K = 1$ in the cases of accuracy, precision, recall, and F1-score and where K varies in MAP

and accuracy @ K. Obtaining the IR scores for the models makes it possible to assess whether or not the vector representations generated by the models enable for search and retrieval as per research question 1 in section 1.2.

Introducing labels for the similarity-searching experiments also means that it is possible to introduce a naive baseline method. This baseline is based on random guessing of the labels without any idea of the vector representation of the input. In the cases of superclasses, the random baseline will predict randomly in the labels $\{1, 2, 3, 4, 5\}$. In the cases of subclasses, it will predict randomly in $\{101, 102, ...110, 201, ...510\}$. Random guessing of labels is expected to correctly classify the audio recording 1/5 times in the superclass setup and 1/50 times in the subclass setup. This means that the accuracy of the random baseline is expected to be respectively 0.200 and 0.020 for the two setups after a sufficient amount of samples to evaluate.

Outlining these experiments stresses how the models are going to generate audio vector representations and how they are going to be assessed and compared. Furthermore, this chapter has also traced several subproblems that are interesting to investigate to capture the full picture regarding these two models and their performance in terms of IR scores when searching by similarity. The results of these experiments are described in the next chapter along with a brief objective analysis of these.

CHAPTER 4

Results & Analysis

This chapter will present the outcomes of the experiments described in the previous section. A brief analysis of the outcomes will also follow along with the choices made based on that experiment if any.

The chapter is essentially divided into 3 sections: First, the selection of features in the vector representations for the supervised learning approach is described, followed by the actual similarity-search. Since YAMNet is used as a pre-trained model, this section also including a subproblem of the search problem regarding labels absent in the training data of YAMNet which are present in ESC-50. The next section presents the optimization traces for the training of the VAE along with visual representations of the output of the trained model. The vector representations generated by this model are also used for similarity-search, divided into the super- and subclass problems. Finally, the models will be compared using the methods described earlier in order to answer the research questions in section 1.2.

The IR metrics used are the ones described in section 2.2.1. However, the MAP and the accuracy @ K metrics are extended for several values of K, $\{2, 3, 4, 5\}$, to explore potential relevant retrieval outside of the closest vector in the similarity search. $K = 1$ is intentionally left out as the accuracy @ K where $K = 1$ would be the same as the accuracy and MAP considering 1 item retrieved would always be 1. The rest of the values are chosen to get a better overview of the items retrieved without setting a value of K too high since there are only 5 superclasses and only 40 samples in each subclass.

The similarity-based search and retrieval experiment is evaluated on the vector representations of all 2000 audio recordings since this experiment only includes a spatial comparison and not any model classification. The search is evaluated based on the labels as mentioned in section 3.3 and is conducted as a supervised classification without having the models ever see the labels of the embeddings during training. Note, however, that since YAMNet is trained discriminatively it has seen labels during training. These labels belong to the AudioSet dataset and are not used at any point in this project since YAMNet is used as a transfer learning model.

4.1 Supervised learning approach

4.1.1 Selecting vector representation features

Recall, that the vector representations generated by YAMNet were of different dimensions based on the length of the audio recording that the representation was generated from. Therefore, it was necessary to select certain features in these representations: $\boldsymbol{\mu}$ (mean of the frames in the representation), $\boldsymbol{\sigma}$ (variance of the frames in the representation), and $\boldsymbol{\Delta}$ (a smooth estimate of the first derivative over the bands in the frames in the representation).

The best combinations of respectively $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ and $\boldsymbol{\Delta}$ are presented in table 4.1 and 4.2. All tables for all combinations are available in appendix B. From table 4.1 and 4.2 it is evident that the $\boldsymbol{\Delta}$ data representations is generally the highest scoring data type, only to be surpassed by the $\boldsymbol{\mu}$ data representations in some MAP and accuracy @ K measurements. Since these data representations perform fairly similarly while still being superior to the others it is difficult to argue whether one is better than the other. However, it is expected that the $\boldsymbol{\Delta}$ data features can capture some temporal context as it is based on the first derivative over the signal. This representation of the vector embedding is therefore selected to use for similarity-based searching and retrieval.

4.1.2 PCA & linear classification

The $\boldsymbol{\Delta}$ features are used for the PCA experiment with the reasoning of the previous section. This data is thresholded with values in the range 0.0000 to 0.0025 with a step size of 0.0001. The thresholded data is classified using the multinomial logistic regression and the SVM classifier with a linear kernel. The results of the classifications can be seen in figure 4.1. Figure 4.1 makes it evident that the thresholding of low variance features makes an insignificant gain in accuracy (0.75% for the SVM classifier when thresholding with a value of 0.0002) and even makes the classifications more inaccurate as more features are thresholded away. Therefore will the non-thresholded $\boldsymbol{\Delta}$ data features, i.e. the ”original” $\boldsymbol{\Delta}$ data features be used in the following section for similarity-based searching. The linear classification results of the original $\boldsymbol{\Delta}$ data features with a threshold of 0.0000 are presented in table 4.3. The super- and subclasses accuracies are very close, of respectively 88.75% and 85.75% for the SVM classifier. It therefore only happens relatively few times that the classified label belongs to the same superclass but a different subclass than the label of the representation used to query. This indicates that the vector representations generated by YAMNet are well separated in latent space.

$\boldsymbol{\mu}$	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.882	0.882	0.882	0.882	**0.925**	0.970
Cos. Sim.	0.910	0.911	0.911	0.910	**0.946**	**0.978**

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	**0.936**	0.964	**0.954**	0.952	0.974	0.930
Cos. Sim.	0.954	**0.974**	0.965	**0.966**	0.978	**0.953**

Table 4.1: Information retrieval metrics for the $\boldsymbol{\mu}$ (mean) data features in the representations generated by YAMNet. The following abbreviations are used in the table: `Acc` = Accuracy, `Euc. Dist.` = Euclidian Distance Measure, `Cos. Sim.` = Cosine Similarity Measure. The scores marked in bold represent the highest score across all combinations of data features in the representations. In the case of equal scores with other combinations, both are marked.

$\boldsymbol{\Delta}$	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	**0.888**	**0.889**	**0.888**	**0.889**	0.924	**0.974**
Cos. Sim.	**0.913**	**0.913**	**0.913**	**0.913**	0.945	**0.978**

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	**0.936**	**0.968**	0.951	**0.959**	**0.976**	**0.932**
Cos. Sim.	**0.955**	**0.974**	**0.967**	0.964	**0.980**	0.951

Table 4.2: Information retrieval metrics for the $\boldsymbol{\Delta}$ (delta) data features in the representations generated by YAMNet. The following abbreviations are used in the table: `Acc` = Accuracy, `Euc. Dist.` = Euclidian Distance Measure, `Cos. Sim.` = Cosine Similarity Measure. The scores marked in bold represent the highest score across all combinations of data features in the representations. In the case of equal scores with other combinations, both are marked.

4.1.3 Similarity-based searching

The similarity-based searching and retrieval will use the latent embeddings generated by YAMNet, where it is expected that similar inputs will have representations that are close in latent space. Therefore, the search is carried out as a closest-vector search, where the prediction of the classifier is the label of the vector that is closest to the input representation in latent space. As mentioned in section 4.1.1, the vector embedding representation used will be the $\boldsymbol{\Delta}$ data features. An example of two closest vectors is seen in figure 4.2. The predictions, based on the two different measures, are used to calculate IR metrics for the experiment. These metrics are shown in table 4.2.

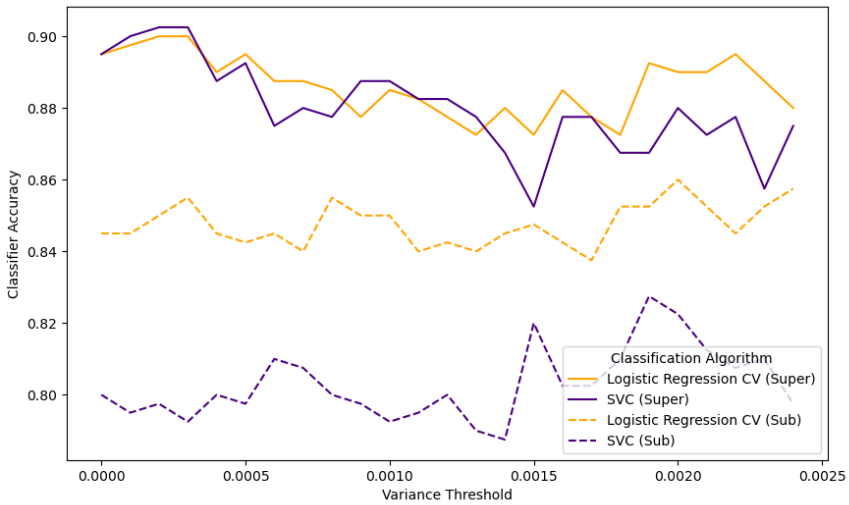

Figure 4.1: The accuracies for the linear classifiers in the latent space for classification of superclasses (super) and subclasses (sub). Note the small scale in the accuracy on the y-axis, pointing towards very little variance in accuracy when thresholding. A total maximum gain of 0.75% accuracy is obtained with the thresholding, meaning that the quality of the PCA does not improve with thresholding features contributing to low variance.

	Multinomial Logistic Regression (5-fold)	SVM Classifier
Superclasses	0.8875	0.8875
Subclasses	0.8475	0.8575

Table 4.3: Linear classification results on the original $\mathbf{\Delta}$ data features from the vector representations generated by YAMNet.

Along the metrics, a confusion matrix for the $\mathbf{\Delta}$ data features are also provided in figure 4.3 to provide a visual representation of the searching experiment.

4.1.3.1 Missing YAMNet labels

As a subproblem in search and retrieval, it was investigated whether or not the non-overlapping classes in ESC-50 and AudioSet would influence the performance when searching by similarity. These labels are described in section 3.1.5. The bar plots in figure 4.4 show the classification score for the 5 missing labels (top row) for each

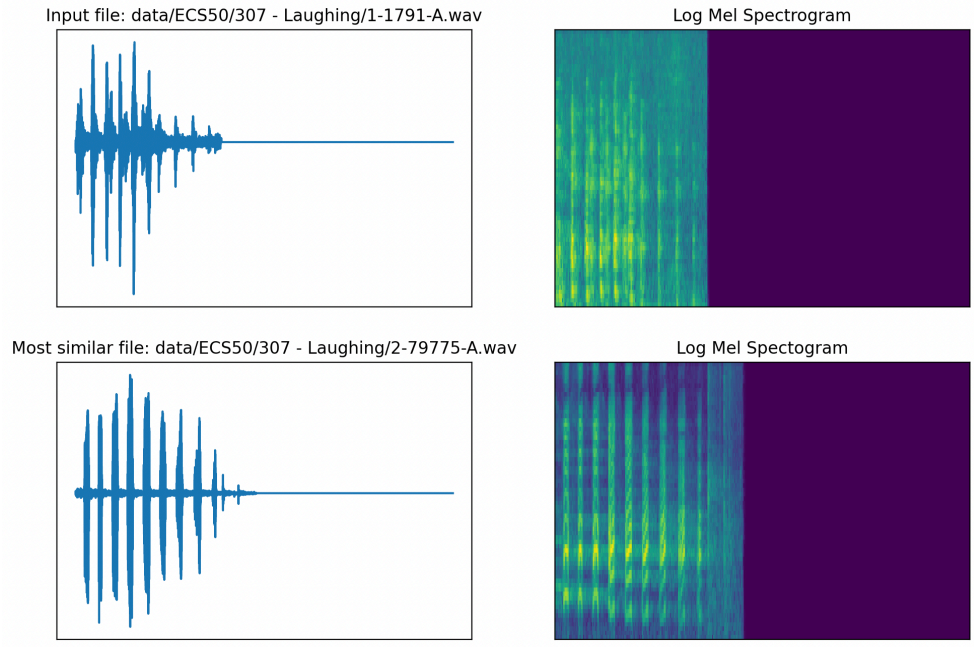

Figure 4.2: An example of a relevant item retrieved by similarity searching in the embeddings generated by YAMNet. Here, the query audio recording `307 - Laughing/1-1791-A.wav` retrieved the recording `307 - Laughing/2-79775-A.wav`. Since both the super- and subclass labels of the query recording and the retrieved recording are the same, this would count as a correct prediction in the searching experiment.

of the two distance measures (Euc = Euclidian Distance, Cos = Cosine Similarity) compared to 5 random classes from the dataset that are present in both ESC-50 and AudioSet. From these bar plots, it is not evident that there should be any difference in the accuracy when searching by similarity with the $\boldsymbol{\Delta}$ features.

4.2 Unsupervised learning approach

4.2.1 Optimization traces

The VAE model was trained on an NVIDIA A100-PCIE-40GB GPU, accessed through the High Performance Computing (HPC) service at DTU. Running K-fold CV with 5 folds, each of 1000 epochs, experiments with different dimensionality in the latent space yielded the optimization traces in figure 4.5. From these optimization traces[1],

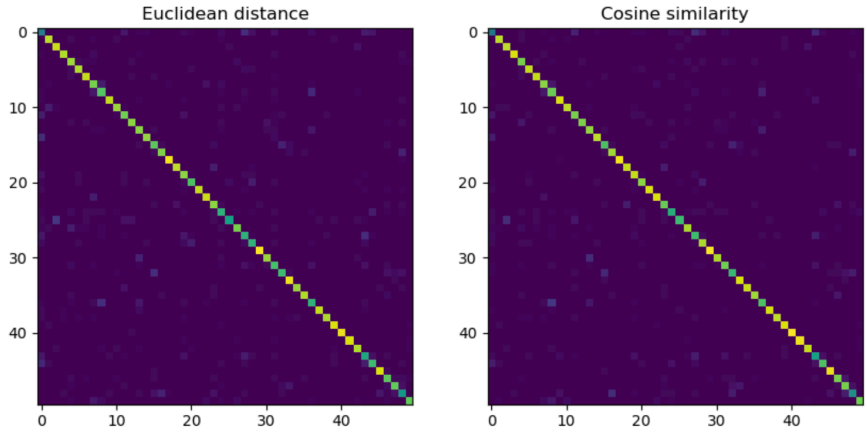

Figure 4.3: Confusion matrices for the retrieved items in the similarity search using the Δ (delta) data features of the embeddings generated by YAMNet. Here, the subclass labels are depicted (50 classes). The brighter yellow a cell is in this matrix, the more correct predictions (i.e. items retrieved with the same subclass label as the query embedding) the search yielded.

it is evident that the model with latent dimensionality of 128 generates the lowest loss in training. This model is therefore used for reconstructions in section 4.2.2 as well as for generating the embeddings for linear classification in section 4.2.3.

4.2.2 Reconstructions

Since the combination of a latent layer and a probabilistic decoder allows the VAE to act like a generative model, the output of the architecture can be a reconstruction of the input based on the latent representation. By using the mean values of the pixels in the last convolutional layer as the reconstruction, it is possible to visualize how well the model is able to reconstruct the input spectrogram based on its latent representation of it.

An example of 5 reconstructions is shown in figure 4.6. These reconstructions are created by the trained model with latent dimensionality of 128. The top row contains 5 original spectrograms belonging to the validation set (i.e. previously unseen by the model). The image right below each original image is the reconstruction of the input. This reconstruction is a direct visualization of the mean output layer when input is passed through a trained model.

[1]The optimization traces for the experiments are also publicly available here: https://api.wandb.ai/links/magnusgp/jkaoxs8l

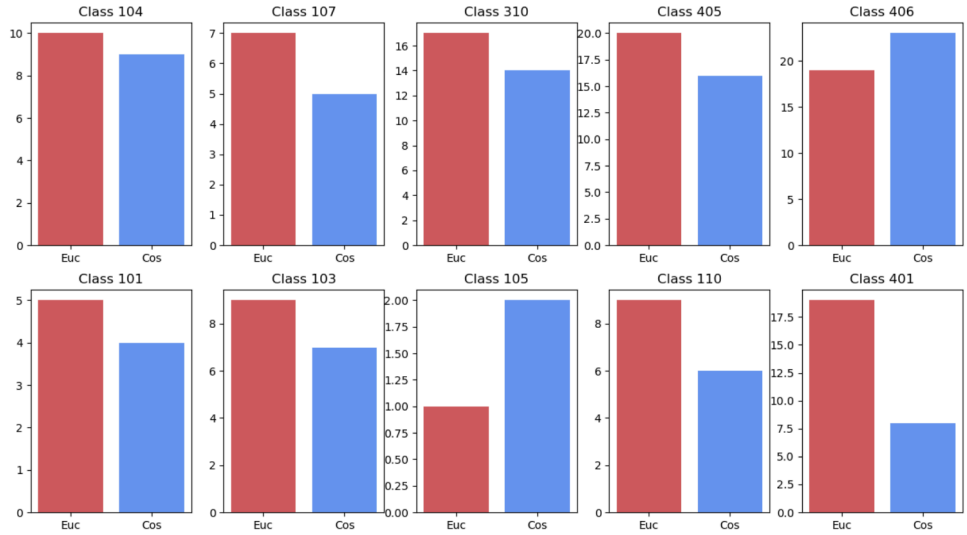

Figure 4.4: Comparison of similarity-search accuracy between the 5 labels present in ESC-50 and not in AudioSet (top row, labels $104, 107, 310, 405, 406$) and 5 random labels present in both datasets (bottom row, labels $101, 103, 105, 110, 401$). This similarity search was conducted using the subclass labels. Abbreviations: Euc = Euclidian Distance, Cos = Cosine Similarity.

4.2.3 Linear classification of classes based on latent embeddings

When considering a similarity-searching experiment, it is necessary that the vector representations are both clustered well together in the latent space as well as separated wrt. classes so that vectors belonging to different classes will not be closest in the similarity search, essentially causing the query to retrieve a non-relevant item. Therefore, it was proposed to fit linear classifiers in the latent space to assess whether or not these vectors were separated.

Fitting a multinomial logistic regression model on the latent vectors yielded the results presented in table 4.4. For superclass classification, the mean accuracy was 0.423 and for subclass classification, the mean accuracy was 0.263. A multinomial SVM classifier was also fitted but did not produce higher accuracies. While the accuracies are not tremendously high, they still seem to indicate some sort of separation in the latent space since they are above a random accuracy of 0.200 for superclasses and 0.020 for subclasses.

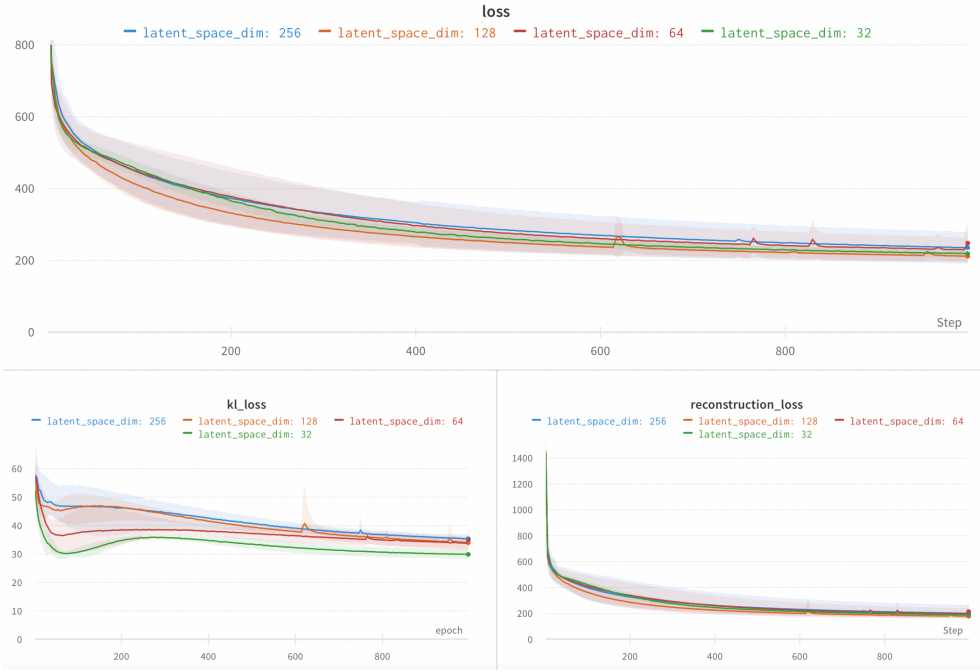

Figure 4.5: Optimization traces for 4 different runs, each with a different dimensionality in the bottleneck layer. The traces are a mean of the loss in a 5-fold CV, with the variance across all 5 runs added as a band in the same color as the trace. The y-axis in each figure represents respectively the loss, the D_{KL} loss, and the reconstruction loss during training. The x-axis is the number of epochs trained, ending at 1000 epochs. The optimization traces are publicly available here: https://api.wandb.ai/links/magnusgp/jkaoxs8l.

4.2.4 Similarity-based searching

Searching in the latent space with respect to each embedding was carried out as outlined in the pseudocode in algorithm 2. This similarity-based searching is conducted without having the model ever see the labels of the embeddings, however, these labels are still used when assessing the performance of the model. As mentioned earlier, a correct classification is one where the vector closest to the query vector and the query vector itself shares the same label.

Table 4.5 presents the IR measures of the best-performing model, i.e. the one with the highest IR scores, namely the one trained with a latent dimensionality of 64. The IR measures for the rest of the runs are presented in appendix C. Even though table

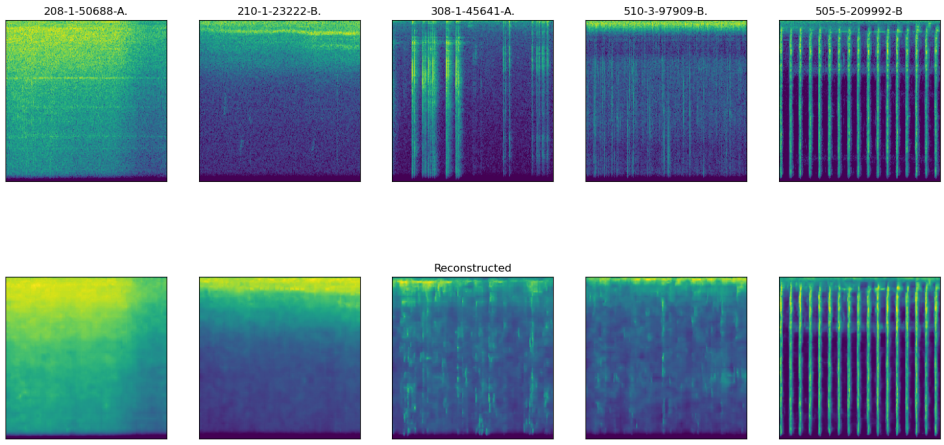

Figure 4.6: Reconstructions using the model with a latent dimensionality of 128. This model performed the best in regards to IR when compared to the other dimensionalities of $\{32, 64, 256\}$. Visualizations of reconstructions of all models are available in appendix D.

	Multinomial Logistic Regression (5-fold)	SVM Classifier
Superclasses	0.423	0.400
Subclasses	0.263	0.123

Table 4.4: Linear classification results on the latent embeddings generated by VAE with latent dimensionality 128.

4.5 presents the results of the most accurate model, these results are still very close to the naive random baseline method introduced at the beginning of the chapter. The reasons behind this are discussed in chapter 5.

4.3 Comparisons

The performance of the two models used in this project is based on both the IR metrics as described in section 2.2.1 as well as the statistical a McNemar test as described in section 2.2.2. The purpose of this comparison is to assess the second research question regarding whether the latent representations will be better for querying a collection of audio recordings when generated in a supervised manner versus an unsupervised manner.

Subclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.0255	0.026	0.026	0.025	0.519	0.08
Cos. Sim.	0.0205	0.02	0.021	0.02	0.476	0.083
	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.578	0.07	0.69	0.052	0.831	0.038
Cos. Sim.	0.537	0.07	0.643	0.052	0.785	0.036
Superclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.205	0.205	0.205	0.205	0.565	0.559
Cos. Sim.	0.19	0.19	0.19	0.189	0.554	0.553
	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.616	0.503	0.696	0.424	0.813	0.327
Cos. Sim.	0.607	0.492	0.681	0.416	0.799	0.317

Table 4.5: IR scores of the best-performing model with latent dimensionality: 64. These scores are essentially equal to the scores of the random baseline (0.020 for subclasses and 0.200 for superclasses). This means that the relevant items retrieved when searching by similarity using the embeddings generated by the VAE can be retrieved just as well by retrieving randomly. Essentially, this deems the embeddings by the VAE useless when searching by similarity with Euclidean distance and cosine similarity.

As noted previously, YAMNet is used as a pre-trained model which has been trained on the very large AudioSet dataset. This means that these comparisons might not be entirely fair due to the large difference in training data. While this is important to keep in mind in the next section, the comparisons are still performed in order to answer research question 2 from section 1.2.

4.3.1 IR

The comparison of the best-performing variant of the VAE model with the best latent combinations of the YAMNet embeddings can be seen in table 4.6. From table 4.6 it is clear that YAMNet outperforms VAE tremendously across all IR terms, both when similarity-searching in superclasses as well as subclasses. This indicates that it either is the case that the embeddings generated by the VAE do not capture the features in the audio well or that the measurements used (euclidean distance and cosine similarity) does not work well for a similarity-based search in these embeddings. This reasoning, along with other factors such as the fact that YAMNet is trained on a lot more data than the VAE will be discussed further in chapter 5.

Superclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
YAMNet						
Euc. Dist.	0.888	0.889	0.888	0.889	0.924	0.974
Cos. Sim.	0.913	0.913	0.913	0.913	0.945	0.978
VAE						
Euc. Dist.	0.205	0.205	0.205	0.205	0.565	0.559
Cos. Sim.	0.19	0.190	0.190	0.189	0.554	0.553

Subclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
YAMNet						
Euc. Dist.	0.812	0.797	0.786	0.788	0.896	0.899
Cos. Sim.	0.838	0.824	0.820	0.82	0.906	0.931
VAE						
Euc. Dist.	0.026	0.026	0.026	0.025	0.519	0.080
Cos. Sim.	0.021	0.020	0.021	0.02	0.476	0.083

Table 4.6: Selected IR metrics for the similarity search of the latent embeddings from the two models based on respectively super- and subclasses. Note, that the VAE model referenced here has a latent dimensionality of 64 since this was the highest-scoring version of the VAE runs.

4.3.2 McNemar's test for model comparison

A McNemar test is based on the contingency table for the two models as described in section 2.2.2. The contingency tables for respectively the super- and subclass problems are shown in tables 4.7 and 4.8.

Both tests for both super- and subclasses provide convincing evidence that the null hypothesis $H_0 : p(\text{VAE}) = p(\text{YAMNet})$ is false since $p = 0.000 < \alpha = 0.050$. H_0 is therefore rejected in favor of the alternative hypothesis $H_1 : p(\text{VAE}) \neq p(\text{YAMNet})$. From the contingency tables, it is also evident that YAMNet is consistently more correct than VAE whenever the two models disagree. This analysis leads to the conclusion that the embeddings generated by YAMNet in this project are better for search and retrieval in terms of IR and a statistical test when compared to the VAE model. However, there is a likelihood that it is the case due to the fact that YAMNet is trained on a lot more data. This imbalance in the amount of training data will be discussed further in section 5.2.1.

To summarize, embeddings generated by both YAMNet and the VAE were used for a similarity searching experiment. The delta features of the YAMNet embeddings were selected for this experiment, which were then compared with the embeddings generated by the VAE model with a latent dimensionality of 64. YAMNet was far superior both in terms of IR scores as well as when being statistically evaluated in a

Euclidean Distance		
	YAMNet Correct	YAMNet Incorrect
VAE Correct	292	52
VAE Incorrect	1472	184
	$p = 0.000$	$\chi^2 = 1321.234$

Cosine Similarity		
	YAMNet Correct	YAMNet Incorrect
VAE Correct	310	33
VAE Incorrect	1511	146
	$p = 0.000$	$\chi^2 = 1412.907$

Table 4.7: Contingency table and results of the McNemar test on similarity search in superclasses for the embeddings generated by the two models.

Euclidean Distance		
	YAMNet Correct	YAMNet Incorrect
VAE Correct	25	3
VAE Incorrect	1523	449
	$p = 0.000$	$\chi^2 = 1512.032$

Cosine Similarity		
	YAMNet Correct	YAMNet Incorrect
VAE Correct	33	3
VAE Incorrect	1615	349
	$p = 0.000$	$\chi^2 = 1604.030$

Table 4.8: Contingency table and results of the McNemar test on similarity search in subclasses for the embeddings generated by the two models.

McNemar test. As mentioned before, it is important to recall here that YAMNet has been trained on a larger dataset.

The visual representations of the outputs of the VAE showed that the model was able to reconstruct a somewhat similar spectrogram from a vector representation of the input and a multinomial linear classification of the vector representations in a high-dimensional hinted a separation of both super- and subclasses in the latent space, however, the embeddings still didn't perform well when used for similarity-searching.

CHAPTER 5

Discussion

The results of the similarity-based search experiments that showed vector representations generated by YAMNet significantly outperformed those generated by VAE, where the latter was only comparable to a random baseline. Intuitively, this leads to the conclusion that a VAE model is not suitable for this specific setup. However, there might be several reasons for this and the most obvious include the differences in model implementation as well as the circumstances of the experiments. These reasons will be discussed in the following sections along with the challenges met during the project.

Lastly, a brief discussion on further perspectives on the research questions is provided.

5.1 Information retrieval and model assessment

The IR metrics described in section 2.2.1 all cover different ways of assessing how well the predictions of the search match the actual class labels of the audio recordings. Since the questions of interest in this project regard whether the models can generate meaningful representations, it has been investigated how much information a query with a vector representation can retrieve.

While the VAE model was able to retrieve some relevant similar vectors when considering its top-k predictions, it still performed worse than random in these cases, deeming the latent embeddings generated from the experiment reported in this project essentially useless for search and retrieval.

YAMNet on the other hand achieved fairly good IR measures in the search experiment along with accuracy @ k-values significantly beating the random baseline. This leads to the belief that the latent embeddings generated by YAMNet can be used for searching and retrieval and actually retrieve the relevant audio recordings.

It is important to assess exactly what IR metrics can be used for when evaluating the query search and where it meets its limitations. First of all, the scores used in this project all measure the relevance of the predictions/retrievals when querying. By combining several metrics, it is possible to evaluate a query based on several factors.

This means, that even though the accuracy of the query may be low it may still have retrieved relevant items in its top-k predictions. This gives a more nuanced picture of the quality of the query. Moreover, the metrics provide scores that can be compared directly across models, since the queries are performed in the same way for both YAMNet and VAE embeddings.

Even though the metrics are comparable, they are only useful as long as there's a label available for evaluation. There are no labels available in the use case provided by WSA and obtaining these would both be challenging and time-consuming for example by human annotation. While the ESC-50 dataset includes labels, these are only used for scoring the IR metrics. These metrics would not suffice in a "real" scenario such as in the WSA and other approaches would be necessary. An approach in this setting could be to conduct user studies where humans would score the retrieved audio recordings according to their subjective belief of relevance. Nonetheless, these studies would again be time-consuming and challenging as human relevance scoring is subjective and costly.

5.2 Model differences

A big part of this project was to assess IR score differences between the two models based on different learning methods. As it is quite evident that these differences are present, it is of interest to compare the experimental setups of the models themselves as well in an attempt to uncover some of the reasons behind such differences.

5.2.1 Model training

A key factor in the difference of model performance may be the data which the two models were trained upon. Since YAMNet is a large transfer learning model, available with pre-trained weights it is hard to compare it directly to the VAE that is trained from scratch. The pre-trained weights are based on training with the large AudioSet dataset, which is not available for public download or usage at the current time of writing. YAMNet is also only publicly available as a transfer learning model and there is not an existing available framework for training the model from scratch on for example the ESC-50 dataset.

As mentioned several times before, YAMNet is trained on the very large AudioSet. Consequently, YAMNet may be able to learn more regularized and robust representations of the audio recordings. AudioSet also includes audio recordings of different quality and length which could lead to YAMNet having learned more general features that are also applicable in the case of a relatively clean dataset as ESC-50. It is also important to note that YAMNet has been trained specifically for the classification of audio based on the vector representations it generates. A VAE model is not trained

specifically for this but is instead a generative model that creates a vector representation as a part of its architectural design which may not be optimal for similarity searching at least in its vanilla form.

Furthermore, it is highly likely that the hyperparameters of YAMNet have been optimized heavily before public release. These hyperparameters are tuned to the task of specifically classifying audio based on an audio vector embedding. The same scope of hyperparameter tuning has not been possible for the VAE model, both due to a restricted timeframe along with computational issues as discussed in section 5.7.

5.2.2 Model architectures & sizes

While both models considered in this project use convolutional layers for dimensionality reduction, their architectures are still very different. With YAMNet using a total of 15 layer blocks, the VAE architecture in this project only has 8. Also, the number of trainable parameters in each model is quite different. The YAMNet model contains 3.729.481 trainable parameters while the VAE model with a latent dimensionality of 128 contains 39.557.490 parameters. A lot of the parameters in the VAE are due to the decoder part of the model, which while being necessary for training, is not used when generating the latent representations of an input.

The YAMNet architecture does not have a decoder-like structure but instead outputs class probabilities and does not suffer from the heavy addition in parameters. In fact, the encoder part of the VAE only contains 4.674.992 of the total trainable parameters, a number which is comparable to YAMNet. Nonetheless, it seems like the depthwise separable layers enable YAMNet to perform better under fewer trainable parameters, as it was also the conclusion in the original MobileNet paper [34]. Using depthwise separable layers as a replacement for traditional convolutional layers in the VAE could therefore serve as an interesting investigation for future research.

5.2.3 Approximations in the VAE

It is also worth recalling that we make a few approximations in the VAE model setup. First of all, we introduce the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$, which of course only can approximate the true posterior to a certain extent. Furthermore, since the ELBO only bound the true likelihood it is another approximation introduced in the model. These factors could account for some of the suboptimal performance, a problem discussed by e.g. Cremer et al [52] and Hjelm et al [53].

5.3 Latent space dimensionality

Since the comparison of the two learning methods is based on their similarity-based search results, it is important to take the background of these results into account. It was evident that YAMNet had a significantly higher accuracy when classifying, pointing towards a better representation in the latent space. However, the dimensionality of the two different latent spaces was not exactly equal and this is very likely a reason why the models performed very differently.

Recall, that the final dimensionality of the YAMNet model was 2048, compared to the latent dimensionality of 64 of the best performing VAE model. Being able to store the audio features in a latent space with a dimensionality many times bigger, it could be expected that the larger dimensionality would be more suitable for representing the features. However, from the search and retrieval results, it is concluded that a lower dimensional latent space actually outperformed higher dimensionality of 128 and 256. Nonetheless, the dimensionality of the latent space can be seen as an optimization problem and it may not even be the same optimal dimensionality for both models since they map the input to the latent space through different model architectures.

5.3.1 Vector representations

The concept of using vector representations of the audio recordings is based on the assumption that it is possible to capture the most important features in the audio. As it lies in the word, these vectors are mere representations and may suffer from information loss during the encoding process. Some differences in environmental audio recordings can be very subtle while other recordings may suffer from a lot of noise. Compressing the audio into a low dimension may lead to more prominent features being learned while disregarding the subtle ones. This encoding loss may be even greater with a suboptimal hyperparameter choice, which may be the case in the VAE model as discussed in section 5.2.1.

5.3.2 Distance measurements

Another plausible explanation to why the VAE performs with IR scores at a level with the random baseline could be that the distance measurements are not suitable for similarity searching using the representations generated by the VAE. Since the VAE captures features into a space of lower dimensionality comparet to YAMNet and also uses these representations for reconstructions, it may be the case that other similarity-based searching methods would see an increase in IR. These measures could for example be the binary hashing method Locality-Sensitive Hashing (LSH) [54] or path-based graph indexing [55], both of which could be interesting approaches for future research.

5.4 Pixel value distributions

The usage of prior distributions is a very fundamental part of Bayesian inference and the performance of methods based on this foundation relies heavily on whether or not this prior assumption suits the problem. Models in the VAE domain typically use a multivariate Gaussian distribution as the prior since this distribution is fairly simple, tractable, has a wide range of applications, and also makes it possible to derive a simpler estimate of the ELBO as shown earlier. However, it may be the case in practice that the actual likelihood of the data does not follow the prior assumption of a multivariate Gaussian. This is the case in this project, where input data has many pixels with value $x_i \approx 0$ (the reason for this approximation is that there are pixel low values present. For example $x_i = 0.8 \cdot 10^{-8}$, which is most likely due to either normalization or the spectrogram conversion). This means that the prior assumption does not exactly match the input likelihood and therefore raises the question of what implications this may have on the performance of the model. The likelihood of the pixel values in the spectrograms generated from the ESC-50 dataset is shown in figure 5.1. It is very important to note that assuming a Gaussian prior distribution does not necessarily mean that the input likelihood should be exactly the same. By introducing a prior distribution in a VAE, it is possible for the model to regularize the latent space and provide a distribution that allows sampling from all parts of the latent space.

However, if the likelihood of the input deviates significantly from the prior, the model may have some implications when decoding the learned latent space. These implications rise when a learned latent space doesn't capture the entire complexity of the data due to a prior assumption that does not allow this. Since the objective function of the VAE is essentially a trade-off between the reconstruction loss and the regularization term, a large deviation in the likelihood from the prior may cause the VAE to embed representations in a suboptimal latent space, biased towards generating reconstructions more likely under the input likelihood rather than under the prior assumption. Consider the log-likelihood term in the ELBO objective function:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] \tag{5.1}$$

In the case where the likelihood contains a lot of values of $x_i \approx 0, \mathbf{x} \in \mathbb{R}^{(256\times256\times1)}$, the log-likelihood term will be biased towards regions of the space where the input is non-zero due to the prior assumption. This does not seem to be entirely the case for the model, however, since the likelihood distribution of the reconstructions of the validation subset of the data (i.e. data unseen during training) resembles the input distribution in figure 5.1. The distribution over the pixels in the reconstructions can be seen in figure 5.2.

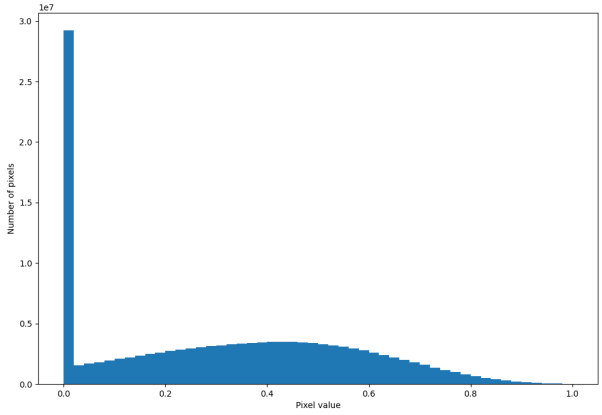

Figure 5.1: The likelihood distribution of the pixel values in the spectrograms generated from the ESC-50 dataset.

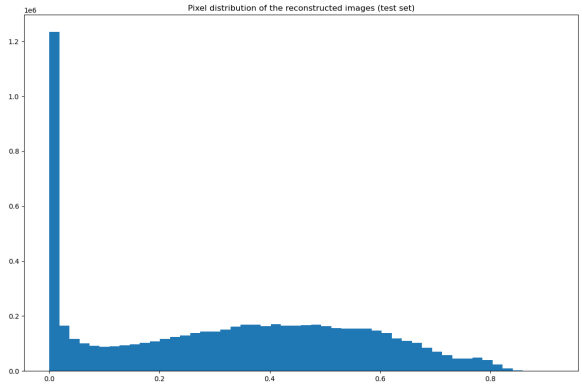

Figure 5.2: The likelihood distribution of the pixel values in the reconstructed spectrograms in the validation dataset. The reconstructions are essentially a visualization of the mean output layer in the VAE. The validation dataset is unseen by the model during training.

5.5 Data complexity

While the ESC-50 dataset had shown fine benchmarking results for environmental audio classification, it still has limitations both in size and complexity. The 2000 recordings over 50 semantic classes leave very few samples per class, which could

cause the model to overfit very quickly. Furthermore, the dataset only consists of 5-second long, relatively noise-free, audio recordings. These recordings can be complex enough in themselves, but if the use case is for larger and noisier recordings it might be hard to apply a model trained on a simple dataset like ESC-50.

The ESC-50 dataset also only consists of environmental audio data. This type of data is of interest in the use case explained in the introduction, however, it is not the only type that WSA wants to explore. It has been sufficient for the scope of this project and has been used to answer the research questions but further research could consider mixing environmental audio with speech, music, and other types of audio to better suit the use case of WSA.

5.5.1 From raw audio to spectrograms

Converting the raw signal into a spectrogram to process it with convolutional layers is a very common approach when combining ML and audio-related tasks. Generating spectrograms over a downsampled signal at 16 kHz could be another case where valuable, but subtle, information could have been lost.

An ideal representation of the raw audio would be to use all data points. Imagine a model that uses a 5-second long clip from the ESC-50 dataset, originally sampled at 44.1 kHz, and feeds it into an input layer capable of taking 220.500 inputs. Then, the model would consist of fully-connected layers, each reducing the dimensionality a bit more until a bottleneck layer is reached. This would potentially pose for a very good vector representation of the input data but the approach also introduces some obvious issues regarding computational issues along with problems regarding overfitting to the data due to the sheer size of this imaginary model.

5.6 Temporal awareness

Audio signals are a digital representation of sound waves through the air. These signals are therefore by nature varying over time and the human interpretation of these depends a lot on temporal contexts. This was the reasoning behind selecting the $\boldsymbol{\Delta}$ features of the YAMNet embeddings over the $\boldsymbol{\mu}$ features, even though the two representations had similar performance in terms of IR. Even though the conversion of the raw signal to a spectrogram keeps some temporal context in terms of the frames, some temporal contexts are still kept since the window size spans several samples. The usage of the GLU activation function also attempts to preserve the temporal context as explained in section 2.5.6

An approach to working with a temporal context in audio signals could be to implement a temporarily aware model, such as the from the LSTM model family. A suggestion for further approach could be to combine the temporal awareness in a

LSTM model with the generative properties of the VAE such as in Bowman et al.: *"Generating Sentences from a Continuous Space"* [56] but for audio data.

5.7 Computational issues

While this project suffers from a lack of usable results for the VAE model, it also met some issues during the project period. First of all, training the VAE models in this project proved to be computationally heavy, with for example 39.557.490 parameters for the model with latent dimensionality of 128. Training this for 1000 epochs per fold for 5 folds averaged around 22 hours per complete run. While it would have been of benefit to the robustness of the model and most likely to the IR scores of it to perform a nested 2-level K-fold CV, this would simply have taken too long to complete in terms of the project deadline.

A lot of the experimental progress in this project came in the last couple of weeks, where a faulty implementation of the likelihood term in the ELBO invalidated previous runs. With this fixed, there was limited time left so model training was performed carefully with great focus on the limited time frame.

Other computational issues met in this project involve the transfer from the local machine to the HPC on DTU. These issues include TensorFlow dependencies, memory shortage, high node queue waiting times, and logging the results in the right way.

5.7.1 β-VAE

The β-VAE setup was implemented as a callback during training of the VAE model to scale the D_{KL} linearly in the first 10 epochs. However, during training, it seemed like this caused a numerical instability in the loss function. This caused the loss values to diverge and essentially result in `nan` values. Therefore, the β-VAE method was disregarded for the experiments and was not considered for evaluation.

5.8 Further Research

Based on the discussed topics in this chapter, further suggested research areas of interest could include:

- Use depthwise separable layers instead of regular convolutional layers in the VAE architecture.
- Experiment with several types of data when training a model for audio-related tasks.

- Combine a the LSTM and VAE approaches for modelling audio data with temporal awareness.
- Explore a more complicated prior for the VAE, for example a Gaussian Mixture Model (GMM).

CHAPTER 6

Conclusion

Motivated by the use case provided from WSA, this project sought to find meaningful ways to capture the important features in environmental audio recordings in a latent space using vector representations. For this, a large-scale pre-trained supervised model, YAMNet, was introduced along with an unsupervised feature-learning model in form of a VAE to generate these vector representations. By conducting a similarity-based search to retrieve relevant items from the same semantic class as the query audio recording it became evident that it was possible to use the vector embeddings to represent the important features in the audio recordings.

The first research question assessed how well an audio vector representation would perform in terms of IR in the search experiment. From the results, it is evident that it is possible to use vector representations of audio for search and retrieval with a superclass accuracy of 91.3% using the YAMNet model to create embeddings of the ESC-50 dataset when the search is conducted with cosine similarity as the distance measure. The same applies to the search in subclasses, here with an accuracy of 83.8% under the same circumstances.

A McNemar test concluded that there was a significant difference in the accuracies of the two models. From the contingency table for both the super- and subclass classification problem, it is concluded that embeddings generated by YAMNet were the source of the more accurate search, most likely because YAMNet had seen a lot more data during its training. This concludes the second research question regarding model comparison, meaning that vector representations generated by the supervised learning method (YAMNet) yielded better IR scores when compared to the unsupervised learning method (VAE).

Further issues and circumstances regarding the models and the experimental setup have also been discussed, with the most notable being the difference in training data and hyperparameter optimization between YAMNet and VAE.

Overall, this project dived into the world of audio processing and explored state-of-the-art methods for audio vector representation. This project can hopefully shed some light on some of the potential possibilities and pitfalls when using audio vector representations for querying a similarity search with both supervised and unsuper-

vised methods.

Suggested further directions of research include using depthwise separable layers inspired by YAMNet and the MobileNet architecture instead of regular convolutional layers in a combined LSTM and VAE architecture as well as using several data types and a more complicated prior in the VAE model.

Acronyms

D_{KL} Kullback-Leibler Divergence 17, 19, 20, 34, 43, 55

.WAV Waveform Audio File 6

Adam Adaptive Moment Estimation 32

AE Autoencoder 1, 16–19

ANN Artificial Neural Network 1

CNN Convolutional Neural Network 2, 3

CNNS Convolutional Neural Networks 1, 3

CV Cross Validation 32, 43, 55

DFT Discrete Fourier Transformation 7

DNN Deep Neural Network 2, 3, 12, 26

ELBO Evidence Lower Bound 17, 19–21, 34, 50, 52, 55

ESC-50 Environmental Sound Classification 50 2, 3, 5, 6, 9, 13, 28, 36, 49, 52–54, 57, 62

FAISS Facebook AI Similarity Search 3

GLU Gated Linear Unit 25, 26, 32, 54

GMM Gaussian Mixture Model 56

HPC High Performance Computing 40, 55

IR Information Retrieval 5, 8, 12, 28, 34–36, 38, 43–46, 48, 49, 51, 54, 55, 57, 63

KNN K Nearest Neighbours 4, 34

LSH Locality-Sensitive Hashing 51

LSTM Long Short-Term Memory 3, 26, 54–56, 58

MAP Mean Average Precision 10, 34, 36, 37

ML Machine Learning 2, 3, 54

PCA Principal Component Analysis 12, 16, 29, 30, 37, 39

ReLU Rectified Linear Unit 26

STFT Short Time Fourier Transformation 7

SVM Support Vector Machine 30, 34, 37, 42

TD-VAE Temporal Difference Variational Autoencoder 4

VAE Variational Autoencoder 1–5, 11–13, 16–19, 21–23, 25, 26, 31–34, 36, 40, 41, 44–53, 55–58, 66

VAEs Variational Autoencoders 2, 26

VQ-VAE-2 Vector-Quantized Variational Autoencoder 2 3

WSA WSAudiology 1, 2, 49, 54, 57

YAMNet `Yet Another Mobile Network` 1–3, 5, 11–16, 28–31, 36–41, 45–51, 54, 57, 58, 63

APPENDIX A

Description of data naming convention

A.1 Naming Conventions

Every recording originally follows the following naming convention [5]:

/{FOLD}-{CLIP_ID}-{TAKE}-{TARGET}.wav

- {FOLD} - The index of the fold used in cross-validation
- {CLIP_ID} - The original ID of the Freesound.org clip
- {TAKE} - Letter in alphabetically ascending order to distinguish different takes of the same Freesound.org clip from each other
- {TARGET} - Class label in numeric format [0;49]

This thesis uses a slightly different naming convention using subdirectories for each subclass, but the conventional parts (for example CLIP_ID and TARGET) are still the same in both conventions:

/{TARGET}-{TARGET_NAME}/{FOLD}-{CLIP_ID}-{TAKE}.wav

- {TARGET} - Class label in numeric format. Classes are divided into hundreds depending on their superclass, i.e. subclasses belonging to superclass 1 will have labels 101, 102, ..., 110, and so on for the other 4 superclasses.
- {TARGET_NAME} - The name of the subclass, for example, ”Dog” or ”Clock Tick”.
- {FOLD} - The index of the fold used in cross-validation
- {CLIP_ID} - The original ID of the Freesound.org clip
- {TAKE} - Letter in alphabetically ascending order to distinguish different takes of the same Freesound.org clip from each other

A.2 Metadata

The following table shows sections of the accompanying metadata of the dataset, here the human classification performance on the audio recordings.

Row Labels	Correct classifications	Total classifications	Accuracy
Airplane	53	78	67,9%
2-105270-A.ogg	1	4	25,0%
2-78781-A.ogg	2	2	100,0%
4-251959-A.ogg	2	2	100,0%
1-11687-A.ogg	1	1	100,0%
1-24796-A.ogg	1	1	100,0%
1-36929-A.ogg	1	1	100,0%
.....	.	.	...
Wind	38	83	45,8%
2-104952-A.ogg	2	6	33,3%
2-109371-C.ogg	1	2	50,0%
5-217186-B.ogg	2	2	100,0%
5-217186-C.ogg	1	2	50,0%
1-137296-A.ogg	1	1	100,0%
1-29532-A.ogg	1	1	100,0%
.....	.	.	...
Grand Total	3203	3939	81,3%

Table A.1: Metadata for the human classification performance on the ESC-50 dataset. The final row represents the grand total. The file extension .ogg represents another way of saving the audio recordings. The content of the recordings is the same.

APPENDIX B

Results of the YAMNet data combination experiment

This appendix shows the full table of every data combination from the YAMNet embeddings. These are presented as IR metrics from the searching experiment based on the latent embeddings generated by the latent layer in YAMNet. The symbols of the data $\boldsymbol{\mu}, \boldsymbol{\sigma}$ and $\boldsymbol{\Delta}$ denote the type of data used. If there's more than one symbol, for example $\boldsymbol{\mu}, \boldsymbol{\sigma}$, then this denotes that the concatenation of these data types is used as the latent representation. These results are summarized in section 4.1.1.

$\boldsymbol{\mu}$	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.882	0.882	0.882	0.882	0.925	0.97
Cos. Sim.	0.91	0.911	0.911	0.91	0.946	0.978
	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.936	0.964	0.954	0.952	0.974	0.93
Cos. Sim.	0.954	0.974	0.965	0.966	0.978	0.953

Table B.1: $\boldsymbol{\mu}$ data.

$\boldsymbol{\sigma}$	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.826	0.827	0.826	0.826	0.893	0.958
Cos. Sim.	0.89	0.89	0.89	0.889	0.933	0.967
	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.914	0.944	0.934	0.926	0.961	0.896
Cos. Sim.	0.946	0.96	0.961	0.948	0.977	0.932

Table B.2: $\boldsymbol{\sigma}$ data.

$\boldsymbol{\Delta}$	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.888	0.889	0.888	0.889	0.924	0.974
Cos. Sim.	0.913	0.913	0.913	0.913	0.945	0.978
	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.936	0.968	0.951	0.959	0.976	0.932
Cos. Sim.	0.955	0.974	0.967	0.964	0.98	0.951

Table B.3: $\boldsymbol{\Delta}$ data.

$\boldsymbol{\mu}, \boldsymbol{\sigma}$	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.868	0.867	0.868	0.867	0.914	0.968
Cos. Sim.	0.891	0.891	0.891	0.891	0.935	0.971
	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.929	0.958	0.948	0.942	0.975	0.914
Cos. Sim.	0.945	0.965	0.961	0.95	0.977	0.934

Table B.4: $\boldsymbol{\mu}, \boldsymbol{\sigma}$ data.

$\boldsymbol{\mu}, \boldsymbol{\Delta}$	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.882	0.882	0.882	0.882	0.925	0.97
Cos. Sim.	0.91	0.91	0.91	0.91	0.947	0.977
	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.936	0.964	0.954	0.952	0.974	0.931
Cos. Sim.	0.954	0.974	0.965	0.966	0.977	0.954

Table B.5: $\boldsymbol{\mu}, \boldsymbol{\Delta}$ data.

$\boldsymbol{\sigma}, \boldsymbol{\Delta}$	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.832	0.833	0.832	0.833	0.893	0.966
Cos. Sim.	0.891	0.891	0.891	0.891	0.932	0.966

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.918	0.944	0.938	0.926	0.962	0.901
Cos. Sim.	0.945	0.958	0.96	0.949	0.978	0.932

Table B.6: $\boldsymbol{\sigma}, \boldsymbol{\Delta}$ data.

$\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\Delta}$	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.868	0.867	0.868	0.867	0.915	0.967
Cos. Sim.	0.89	0.89	0.891	0.89	0.935	0.971

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.929	0.958	0.948	0.942	0.975	0.914
Cos. Sim.	0.945	0.965	0.961	0.95	0.977	0.934

Table B.7: $\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\Delta}$ data.

APPENDIX C

VAE search & retrieval

This appendix shows the full tables for the different VAE models with latent dimensions $\{32, 64, 128, 256\}$.

Subclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.0185	0.018	0.019	0.018	0.506	0.074
Cos. Sim.	0.0195	0.02	0.02	0.019	0.499	0.08

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.558	0.063	0.645	0.05	0.743	0.038
Cos. Sim.	0.548	0.07	0.634	0.054	0.747	0.04

Superclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.183	0.182	0.183	0.182	0.547	0.55
Cos. Sim.	0.184	0.183	0.184	0.183	0.55	0.54

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.601	0.487	0.678	0.41	0.791	0.314
Cos. Sim.	0.602	0.479	0.684	0.4	0.802	0.304

Table C.1: Latent dimensionality: 32.

Subclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.205	0.205	0.205	0.205	0.565	0.559
Cos. Sim.	0.1895	0.19	0.19	0.189	0.554	0.553

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.616	0.503	0.696	0.424	0.813	0.327
Cos. Sim.	0.607	0.492	0.681	0.416	0.799	0.317

Superclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.205	0.205	0.205	0.205	0.565	0.559
Cos. Sim.	0.19	0.19	0.19	0.189	0.554	0.553

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.616	0.503	0.696	0.424	0.813	0.327
Cos. Sim.	0.607	0.492	0.681	0.416	0.799	0.317

Table C.2: Latent dimensionality: 64.

Subclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.1875	0.188	0.188	0.187	0.549	0.558
Cos. Sim.	0.177	0.177	0.177	0.177	0.542	0.548

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.603	0.498	0.677	0.418	0.793	0.32
Cos. Sim.	0.59	0.49	0.669	0.41	0.787	0.308

Superclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.188	0.188	0.188	0.187	0.549	0.558
Cos. Sim.	0.177	0.177	0.177	0.177	0.542	0.548

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.603	0.498	0.677	0.418	0.793	0.32
Cos. Sim.	0.59	0.49	0.669	0.41	0.787	0.308

Table C.3: Latent dimensionality: 128.

Subclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.193	0.193	0.193	0.193	0.552	0.576
Cos. Sim.	0.1905	0.191	0.19	0.191	0.548	0.566

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.602	0.514	0.673	0.44	0.788	0.334
Cos. Sim.	0.607	0.496	0.684	0.414	0.804	0.313

Superclasses	Accuracy	Precision	Recall	F1-Score	MAP @ 5	Acc @ 5
Euc. Dist.	0.193	0.193	0.193	0.193	0.552	0.576
Cos. Sim.	0.19	0.191	0.19	0.191	0.548	0.566

	MAP @ 4	Acc @ 4	MAP @ 3	Acc @ 3	MAP @ 2	Acc @ 2
Euc. Dist.	0.602	0.514	0.673	0.44	0.788	0.334
Cos. Sim.	0.607	0.496	0.684	0.414	0.804	0.313

Table C.4: Latent dimensionality: 256.

APPENDIX D

VAE Reconstructions

TODO: Add all reconstructions here

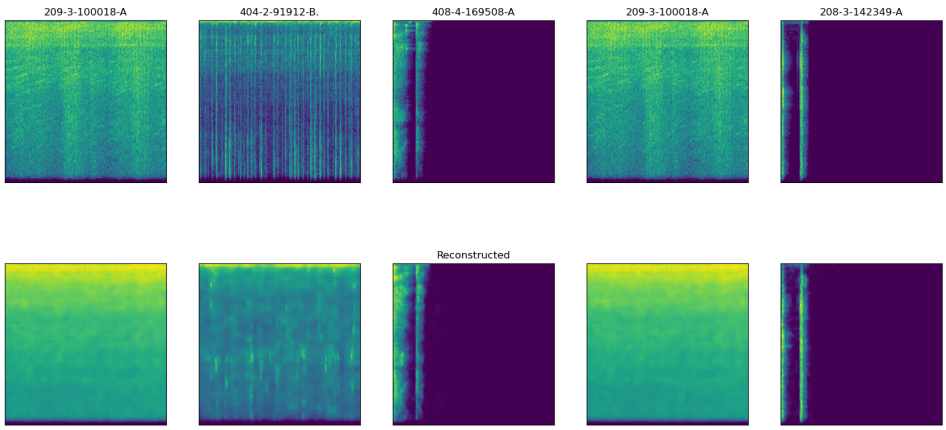

Figure D.1: Reconstructions using the model with a latent dimensionality of 32.

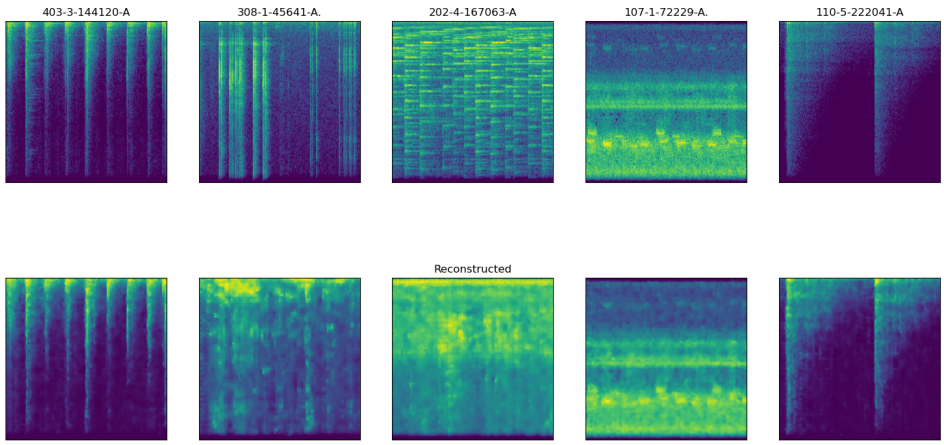

Figure D.2: Reconstructions using the model with a latent dimensionality of 64.

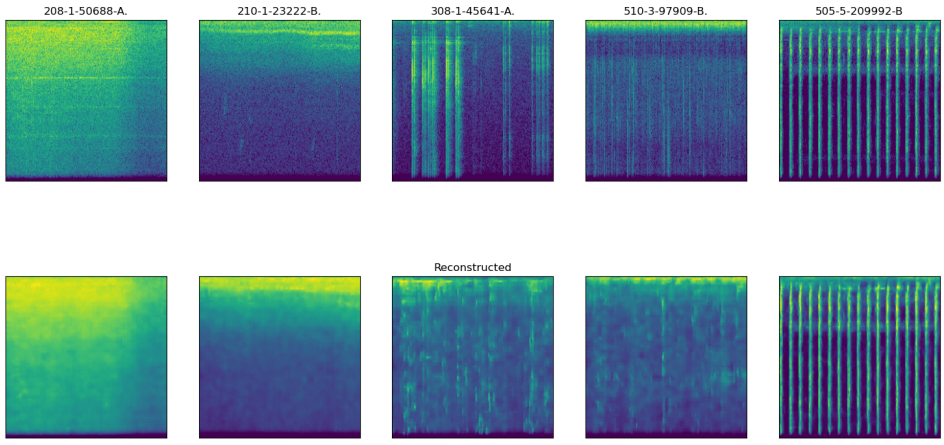

Figure D.3: Reconstructions using the model with a latent dimensionality of 128.

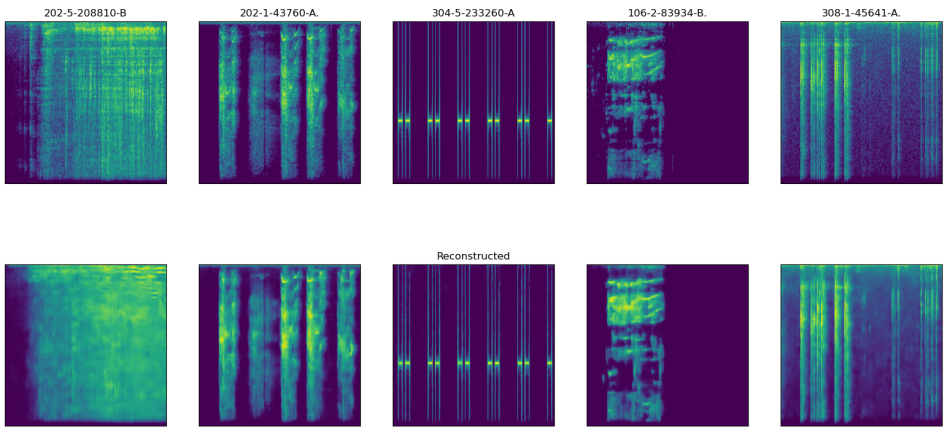

Figure D.4: Reconstructions using the model with a latent dimensionality of 256.

Bibliography

[1] Shawn Hershey et al. “CNN Architectures for Large-Scale Audio Classification” (September 2016).

[2] Diederik P Kingma and Max Welling. “Auto-Encoding Variational Bayes” (December 2013).

[3] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. *Stochastic Backpropagation and Approximate Inference in Deep Generative Models*. Technical report. 2014.

[4] Manoj Plakal and Dan Ellis. *YAMNet*. September 2021.

[5] Karol J. Piczak. “ESC: Dataset for Environmental Sound Classification.” In: *Proceedings of the 23rd ACM international conference on Multimedia*. New York, NY, USA: ACM, October 2015, pages 1015–1018. ISBN: 9781450334594. DOI: 10.1145/2733373.2806390.

[6] Sanyuan Chen et al. “BEATs: Audio Pre-Training with Acoustic Tokenizers” (December 2022).

[7] Adrià Recasens et al. “Broaden Your Views for Self-Supervised Video Learning” (March 2021).

[8] Calum Heggan et al. “MetaAudio: A Few-Shot Audio Classification Benchmark” (April 2022).

[9] Andrey Guzhov et al. “AudioCLIP: Extending CLIP to Image, Text and Audio” (June 2021).

[10] Ben Gold, Nelson Morgan, and Dan Ellis. “Speech and Audio Signal Processing: Processing and Perception of Speech and Music: Second Edition.” *Speech and Audio Signal Processing: Processing and Perception of Speech and Music: Second Edition* (October 2011). DOI: 10.1002/9781118142882.

[11] Aurelio Uncini. “Digital Audio Processing Fundamentals.” Springer Topics in Signal Processing 21 (2022). DOI: 10.1007/978-3-031-14228-4. URL: https://link.springer.com/10.1007/978-3-031-14228-4.

[12] Hervé A. Bourlard and Nelson Morgan. *Connectionist Speech Recognition*. Boston, MA: Springer US, 1994. ISBN: 978-1-4613-6409-2. DOI: 10.1007/978-1-4615-3210-1.

[13] Abdel-rahman Mohamed, George E. Dahl, and Geoffrey Hinton. "Acoustic Modeling Using Deep Belief Networks." *IEEE Transactions on Audio, Speech, and Language Processing* 20.1 (January 2012), pages 14–22. ISSN: 1558-7916. DOI: 10.1109/TASL.2011.2109382.

[14] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. "A Dataset and Taxonomy for Urban Sound Research." In: *Proceedings of the 22nd ACM international conference on Multimedia*. New York, NY, USA: ACM, November 2014, pages 1041–1044. ISBN: 9781450330633. DOI: 10.1145/2647868.2655045.

[15] Jort F. Gemmeke et al. "Audio Set: An ontology and human-labeled dataset for audio events." In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, March 2017, pages 776–780. ISBN: 978-1-5090-4117-6. DOI: 10.1109/ICASSP.2017.7952261.

[16] Iurii Lezhenin, Natalia Bogach, and Evgeny Pyshkin. "Urban Sound Classification using Long Short-Term Memory Neural Network." In: September 2019, pages 57–60. DOI: 10.15439/2019F185.

[17] Jonghee Sang, Soomyung Park, and Junwoo Lee. "Convolutional Recurrent Neural Networks for Urban Sound Classification Using Raw Waveforms." In: *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, September 2018, pages 2444–2448. ISBN: 978-9-0827-9701-5. DOI: 10.23919/EUSIPCO.2018.8553247.

[18] Soo Hyun Bae, Inkyu Choi, and Nam Soo Kim. *Detection and Classification of Acoustic Scenes and Events*. Technical report. 2016.

[19] Erik Bernhardsson. *Annoy (Approximate Nearest Neighbors Oh Yeah)*. 2023.

[20] Alexei Baevski et al. "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations" (June 2020).

[21] Jeff Johnson, Matthijs Douze, and Hervé Jégou. "Billion-scale similarity search with GPUs" (February 2017).

[22] *https://www.pinecone.io*.

[23] *https://milvus.io*.

[24] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. "Generating Diverse High-Fidelity Images with VQ-VAE-2" (June 2019).

[25] Karol Gregor et al. "Temporal Difference Variational Auto-Encoder" (June 2018).

[26] Frederic Font, Gerard Roma, and Xavier Serra. "Freesound technical demo." In: *Proceedings of the 21st ACM international conference on Multimedia*. New York, NY, USA: ACM, October 2013, pages 411–412. ISBN: 9781450324045. DOI: 10.1145/2502081.2502245.

[27] Brian Mcfee et al. *librosa: Audio and Music Signal Analysis in Python*. Technical report. 2015. URL: https://www.youtube.com/watch?v=MhOdbtPhbLU.

[28] Tom Bäckström et al. *Introduction to Speech Processing*. 2nd edition. https://speechprocessing 2022. URL: https://speechprocessingbook.aalto.fi.

[29] Julius O. Smith. "Spectral Audio Signal Processing." *Center for Computer Research in Music and Acoustics (CCRMA)* (2011), pages 1–674. URL: https://ccrma.stanford.edu/~jos/sasp/%20http://books.w3k.org/.

[30] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. "Evaluation in information retrieval." In: *Introduction to Information Retrieval*. Cambridge University Press, 2009. Chapter 8, pages 151–175.

[31] Quinn McNemar. "Note on the sampling error of the difference between correlated proportions or percentages." *Psychometrika* 12.2 (June 1947), pages 153–157. ISSN: 00333123. DOI: 10.1007/BF02295996/METRICS. URL: https://link.springer.com/article/10.1007/BF02295996.

[32] Allen L. Edwards. "Note on the "correction for continuity" in testing the significance of the difference between correlated proportions." *Psychometrika* 13.3 (September 1948), pages 185–187. ISSN: 00333123. DOI: 10.1007/BF02289261.

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, pages 499–523. URL: http://www.deeplearningbook.org/%20https://www.deeplearningbook.org/contents/autoencoders.html.

[34] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications" (April 2017).

[35] Ho-Hsiang Wu et al. "wav2CLIP: Learning robust audio representations from CLIP" (2021). URL: http://dcase.community/challenge2020/task-automatic-audio-captioning.

[36] Delwar Hossain et al. "Real-Time Food Intake Monitoring Using Wearable Egocnetric Camera." In: *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, July 2020, pages 4191–4195. ISBN: 978-1-7281-1990-8. DOI: 10.1109/EMBC44109.2020.9175497.

[37] Alberto Tena, Francesc Clarià, and Francesc Solsona. "Automated detection of COVID-19 cough." *Biomedical Signal Processing and Control* 71 (January 2022), page 103175. ISSN: 17468094. DOI: 10.1016/j.bspc.2021.103175.

[38] Ievgeniia Kuzminykh et al. "Audio Interval Retrieval using Convolutional Neural Networks." *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12525 LNCS (September 2021), pages 229–240. DOI: 10.1007/978-3-030-65726-0{_}21. URL: http://arxiv.org/abs/2109.09906%20http://dx.doi.org/10.1007/978-3-030-65726-0_21.

[39] Abraham. Savitzky and M. J. E. Golay. "Smoothing and Differentiation of Data by Simplified Least Squares Procedures." *Analytical Chemistry* 36.8 (July 1964), pages 1627–1639. ISSN: 0003-2700. DOI: 10.1021/ac60214a047.

[40] Thomas M. Cover. "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition." *IEEE Transactions on Electronic Computers* EC-14.3 (1965), pages 326–334. ISSN: 03677508. DOI: 10.1109/PGEC.1965.264137.

[41] Frank Samuelson and David G. Brown. "Application of Cover's theorem to the evaluation of the performance of CI observers." *Proceedings of the International Joint Conference on Neural Networks* (2011), pages 1020–1026. DOI: 10.1109/IJCNN.2011.6033334.

[42] Jakub M. Tomczak. *Deep Generative Modeling*. Cham: Springer International Publishing, 2022. ISBN: 978-3-030-93157-5. DOI: 10.1007/978-3-030-93158-2.

[43] Shakir Mohamed et al. "Monte Carlo Gradient Estimation in Machine Learning." *Journal of Machine Learning Research* 21 (July 2021), pages 1–62.

[44] Yann N Dauphin et al. "Language Modeling with Gated Convolutional Networks" (2017).

[45] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" (February 2015).

[46] plakal. *Fine tuning Yamnet model, Issue #8425, Comment 622233769*. May 2020.

[47] Mia Hubert et al. "Sparse PCA for High-Dimensional Data With Outliers." *Technometrics* 58.4 (October 2016), pages 424–434. ISSN: 0040-1706. DOI: 10.1080/00401706.2015.1093962.

[48] *sklearn.feature_selection.VarianceThreshold — scikit-learn 1.2.2 documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html.

[49] Diederik P Kingma and Jimmy Lei Ba. "Adam: A Method for Stochastic Optimization" (2014).

[50] Nitish Shirish Keskar et al. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima" (2016).

[51] Casper Kaae Sønderby et al. "How to Train Deep Variational Autoencoders and Probabilistic Ladder Networks." *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016)* (February 2016).

[52] Chris Cremer, Xuechen Li, and David Duvenaud. "Inference Suboptimality in Variational Autoencoders" (January 2018).

[53] R Devon Hjelm et al. "Iterative Refinement of the Approximate Posterior for Directed Belief Networks" (November 2015).

[54] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. "Similarity Search in High Dimensions via Hashing" (1999).

[55] Olivier Siohan and Michiel Bacchiani. "Fast Vocabulary-Independent Audio Search Using Path-Based Graph Indexing" (2005). DOI: 10.21437/Interspeech.2005-52.

[56] Samuel R Bowman et al. “Generating Sentences from a Continuous Space” (2015).