POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Politecnico
di Torino

Master's Degree Thesis

Can you hear what I've learned?

Explaining audio transformer-based models through embedding sonification

Supervisors

Prof. Eliana PASTOR

Prof. Alkis KOUDOUNAS

Candidate

Gabriele TOMATIS

April 2025

Summary

Since their introduction, transformer models showed instantly their high performances in the analysis of structured data such as images, time series and audios. Their ability in solving the most different tasks brought them to become rapidly the state of the art in a wide variety of domains. How they reason, however, is still a big issue, as they translate those data into an embedding representation that only they could comprehend.

Despite that, only a few works try to solve this problem by using several methods proposed in the field of Explainable AI. The aim of this discipline is to make AI models interpretable in a way that makes them trustworthy and reliable; this would be impossible to obtain if we do not understand the way the models reason.

To address these issues, we take advantage of Descript Audio VAE, a model specifically trained to compress and reconstruct an audio waveform passing through a latent space representation. In particular, we apply a gating layer between the embedding space of the model that we want to interpret and the latent space of Descript Audio VAE. This mapping converts the embeddings of the first audio transformer to the latents of the second model that can translate this unknown representation into sound: something that we can interpret.

Acknowledgements

A mamma, papà, mimi e Marco, senza di loro tutto questo non sarebbe stato possibile.
Ai professori, dai quali ho potuto imparare un metodo efficace e che mi hanno saputo guidare attraverso le incertezze e le incomprensioni.
Ad Alsesio, Cla e Stura, per aver reso questi anni universitari più leggeri, divertenti e meno rigorosi.
Ai nervosi, ai pirati, ai biricchini e agli ignoranti, perchè il 30% è ciò che sei tu, il 70% l'ambiente in cui cresci.
Alla musica, eterna colonna sonora che mi ha insegnato che nulla è impossibile, basta avere le idee chiare e saltare.
A tutti coloro che ho e non ho citato finora, perchè, tramite le loro azioni ed il proprio modo di essere, nel bene e nel male, costituiscono un esempio da imitare o evitare, ma da cui, comunque, si può sempre imparare.
GRAZIE!:D

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Transformers are deep learning models commonly used in a wide range of fields, from finance to medicine, from industry to economy. They became rapidly the state of the art for entire research fields like image analysis, temporal sequence predictors or even generative models.

Their main characteristic is attention, which has distinguished them from previous architectures since their inception and allowed them to have a contextual understanding of the input.

To name a few of the most famous examples, think about LLaMA from Meta AI [1], or BERT developed by Google AI [2]. Specifically, the mentioned models are distinguished by being LLMs: Large Language Models capable of interpreting what is said to them with human language and providing results of any kind (text, audio, images) that are better suited to the context. The attention mechanism allows them to achieve exactly this, to extract the meaning from one or more words forming concepts within one or more sentences, giving them the ability to extrapolate some context from what they process.

These large models, that are not limited only on elaborating words, but are also dominant with images, audios or temporal data, consist of millions or billions of parameters and they have changed deeply the world of artificial intelligence. Each big tech company has entered the race to have the best-performing model, optimizing the computational cost of the model, the time and the resources required for training, and the precision of the results. However, the game is not reserved only for big companies; anyone can create their own transformer (but other models too!), starting from an existing model that can be fine-tuned for a specific task different from the original one, or create a custom one that addresses, from the early stages of its development, to one or more specific requests.

Their ability to extract context features leads them to achieve remarkable results, and this has made them rise to the top of popularity rankings for models used in research, providing a solid starting point for various types of analysis. However,

some big issues still exist. One of them is that these models operate in a **black-box** manner. Due to their complexity, they resemble a black box where an input is entered in the model and an output is computed. This process mixes input elements in a way that make unfeasible any sort of connection between the initial and final state. The causes of this opaque behavior are mainly two: in fact, transformer cannot understand human perceptions like sounds, words or images as we intend. It must "translate" them in a way that only itself can comprehend forming what is called the **embedding** representation or the **hidden state** representation. The second issue is due to the high number of model parameters and the complexity of the computation made by itself. Some operations are probabilistic instead of being deterministic, meaning that the computation in one way is easier to obtain than the reversed procedure, that could be obtained despite some approximations. Due to these circumstances, how it decides which response to provide for a given sentence or which parts of an image or audio it focuses on when classifying a sample remain largely unknown.

This is where the research field called **Explainable AI** comes into play, which seeks to provide explanations for the reasons why a particular model behaves the way it does. Given the application task of these new models, it is crucial to trust their decisions. But how can we trust it if the reasoning behind its conclusions is not interpretable and remains unknown to us?

1.1 Transformer architectures

In order to comprehend how transformer models works, let us focus on their general architecture.

This type of model was introduced in the paper by Ashish Vaswani: *Attention Is All You Need* [3], and, since then, it has spread widely across AI models. Its widespread use is due to its distinctive feature: the **attention mechanism**, which enabled the achievement of results that were never seen in previous architectures. Attention is fundamentally a function that allows the model to focus on the context of the input. It is therefore capable of identifying features not only specific to a given input but also the relationships between different parts of it, allowing it to extrapolate concepts as humans understand them as set of different aspects.

As we can see from the picture 1.1 this is a so reasonable way of "thinking". Through attention, the model learns that the order of some words changes during the translation from one language to another and it is able to apply this concept during English-French translation in this case.

The computation of this peculiar characteristic follows the method shown in Figure 1.2.

If we look at Figure 1.2, we can see Q, K and V. These stands for Query, Key and

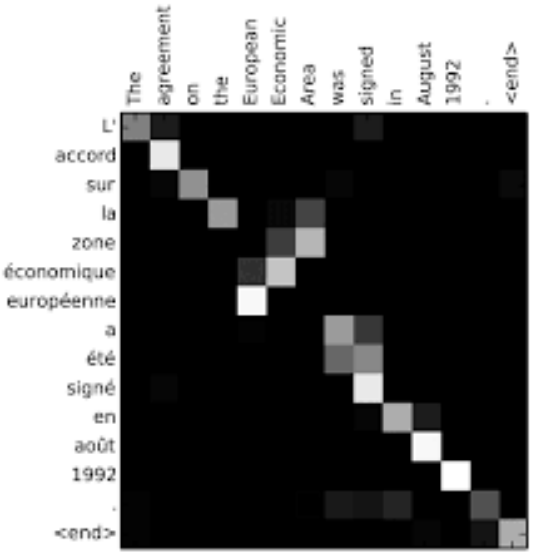

Figure 1.1: Attention mechanism; *Source: Bahdanau et al. "Neural machine translation by jointly learning to align and translate", Neurips 2015*

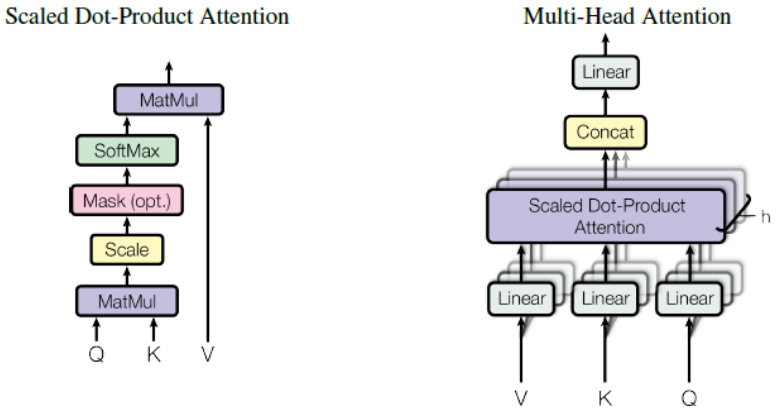

Figure 1.2: Attention computation; *Source: https://www.baeldung.com/cs/attention-mechanism-transformers*

Values and they are usually vectors of a fixed shape that come from the hidden state representation of the model. We can also see how they are combined to produce the

so called self-attention. Firstly, a matrix multiplication computes the dot product between Q and K. Then the output is scaled by dividing it by a certain quantity; then it could be masked. This optional operation randomly zeros some vectors. This aims to introduce more randomness into the computation; this aspect, in fact, help the model to generalize what it learns through the different training epochs. Finally, an activation function prevents that the output values become way too low or high, which could lead to the vanishing or exploding gradient phenomena. The last step multiplies this output by the values that yielded to the attention output. The process described here is called scale dot product attention is computed by one or more attention head. On the right of Figure 1.2 we can see more attention heads working in parallel, this allow to any of them to focus on a particular aspect of the processed input.

Returning to the general description, transformer architecture can consist of two key components: the encoder and the decoder. The encoder is responsible for the input processing, while the decoder handles the decoding of the internal state. The main steps are described here.

Encoder

1. The original input (whether audio, text, or image) is translated into the transformer **embeddings**. Some tokens could be inserted into these hidden states and they are usually exploited to achieve classification results such it happens within AST model.

2. Self-attention is calculated by each attention head that composes the model using the embedding representation by multiplying keys and queries, whose dimensions vary depending on the model, but are typically at least two-dimensional matrices. This results in what is called an attention score(s) or similarity score(s), which represents how strongly different parts of the input are related. This is very useful for focusing on different parts of the same input. The output from each head is then concatenated and becomes the input for the next layer.

Decoder

1. Starting from the encoder representation, the output is generated. Using cross-attention, an aspect that will be clearer soon, the decoder chooses each part of the output (whether words or images) based on the highest probability that a given word belongs to a sentence or a given image being part of a video.

Depending on how these two modules are combined, we have three main transformer architectures:

Encoder only - *e.g. BERT* [2], *AST* [4], *ViT* [5]

In this transformer type, the encoder is responsible for processing the input. This will be then elaborated from other layers such as dense layers and classification heads that are useful when dealing with classification tasks, where the desired output is simply a label related to the input image.

Decoder only - *e.g. LLaMA* [1]

This type of transformer has only the decoder, which is useful for generating output. The output can involve images, audio, or text (videos are considered as sequences of images).

Encoder-Decoder - *e.g. BART* [6]

This architecture is useful for tasks like comprehension and generation, such as text understanding and summarization (sequence-to-sequence models), where both input and output are sequences of text, but they differ from each other.

Depending on the transformer architecture, attention has two main forms: self-attention and cross-attention. ***Self-attention*** refers to attention within the input sequence, enabling the model to interpret sequences of words and process concepts. It is usually used inside the logic of a single component (encoder or decoder), with both input and output of the attention mechanism belonging to the same domain, and can involve multiple parallel heads, each focusing on different aspects of the input sequence. On the other hand, ***cross-attention***, is the mechanism based on two different sequences during the same computation as previously shown in Figure 1.1. Because of this, it typically connects the decoder with the encoder. It is used once within the transformer (no multi-head), and the input and output can belong to different domains (e.g., an image generated from a text prompt).

1.2 The need of explainability

These black-box models are becoming the new state of the art for many different application fields and are actually capable of astonishing results; a big issue that still has to be solved is their explainability and interpretability. **Explainable AI** is a research area that tries to solve these issues to make AI models more understandable for us as humans, in order to better understand their behavior and to increase our trust on them.

Each model, depending on how it was developed, can be explainable by design (think of simpler models like decision trees or classification rules, the so-called white-box models) or not (almost all existing and better performing models are black-boxes). Fortunately, there are some techniques to make even black-box

models interpretable, and this was the main theme of my thesis work.

This research field is also responsible for addressing other aspects, too. If the models were interpretable, in fact, we could easily assess if there is some decision bias in the process, if the procedure is fair or if it is discriminative towards some data labels. Furthermore, also the robustness of the model can be leveraged, allowing us as programmers to build stronger models in terms of adversarial attacks resistance looking also at the privacy policies maintenance. In order to give a brief overview on this discipline, we can explore the taxonomy of this research domain.

We can set five macro-areas that are divided into some smaller ones. Let us see them, first trying to understand what are the main related questions and, later, to analyze them pointwise.

- *Stages of explainability*: at which step the explainability is put into the model?
- *Explanation generalizability*: Is the explanation method valid for possibly every model out there or is it model specific?
- *Scope of explainability*: are the explanations covering all the predictions made by the model? Or are they focused on a single prediction at a time?
- *Explanation representation*: in what form do the explanations assume? For example, audio with higher pitch on focused parts, image with some characteristics highlighted, highlighted text...
- *Methodology used to obtain explanations*: there are several ways to obtain explanations from a model such as explaining by removing, counterfactual explanations or building local surrogate models, but this aspect is so application dependent and may vary a lot for different tasks.

Stages of explainability

The explanations could be placed at different stages:

Pre-modeling explainability focuses on raw data that will be the input of the model, and the aim is to see how the data are organized, visualizing them, and filtering in an efficient way, keeping just the most useful information trying to avoid biases during the computation (this is called *feature engineering*). An important aspect is also to check if the data can be used for our purposes in order to avoid misuse, if they present already known biases (imbalanced dataset) and if there are any ethical considerations.

In modeling explainability finds hints of trustworthiness that are given by the model itself. They are usually simple and *interpretable by design* models. This is the case of decision trees, decision rules, linear and logistic regressions or K-NN classifier. Despite being interpretable, these models are really simple and can solve only a small part of tasks and, because of that, their usage is really limited.

In ***post modeling*** explainability the explanations comes after the model development. Some examples of these techniques are global surrogate models that expect to build a similar simpler and interpretable model with reference to the one that we want to explain; permutation feature importance that randomizes one or two features at a time along their domain, making the features assume all the possible values in their domain. In this way, we can scroll the inter-feature dependency and examine how the loss or the precision metrics vary. So we can determine which are the most important features that are retained to compute the model decision. Finally, the partial dependence plot selects a random features and makes the whole dataset have a single value for that one. Then we make the model predict the samples and we average the various probabilities. We then repeat the procedure for the entire domain of that feature.

Explanation generalizability

The explanation method could be model dependent or model agnostic. In the first case the explanations are valid only for a single model and this gives us the advantage of being more specific, but makes us build another explanation method if we change the model that we want to explain. In the second case, instead, the explanations are valid for every model, but maybe the explanation is local (we will see this definition in the next point) or is computational expensive or maybe it could require additional training data.

Scope of explainability

This aspect is focused on which segment of the model we want to explain. We may want to explain all the predictions made by the model in a comprehensive way, or we may want to define the explanation only for a small group of predictions or going even further and address only one.

Explanation representation

Which form assumes the explanation; they are:

- *Feature importance*: highlighting the part of the input that the model uses the most to assess its prediction.
- *Local rules*: a decision rule for a single prediction.
- *Graph visualization*
- *Explanation by example*: giving one similar sample among the others seen in order to say *"This is x because this is similar to this other one and it is also x."* or *"This is x because this can't be y because the difference is..."*

Methodology used to achieve the explanations

These are the ways that could be employed to retrieve explanations from the model:

- *Explaining by removing*: removing some features to assess how important they are.

- *Local surrogate interpretable models* exploit the interpretability and explainability of simpler models such as decision trees or association rules, but are applied in a local context (one or a few predictions at a time) to provide more useful explanations with respect to a global approach.
- *Gradient-based explanations* use gradient information to provide information on how changes on the input will reverberate on the output.
- *Counterfactual methods* is about generate a similar sample to the one we are going to explain but almost different to be predicted as another class. In this way, we can understand which changes in which features are more important for the decision process.

1.3 Aim of this thesis

After this overview, it is clear that there is a need to fully understand the functioning of these new models, without merely focusing only on their remarkable accuracy. My thesis work focused on audio transformers. Specifically, my work focused on the interpretation of the embedding representation, which is understandable from the model, but completely unknown to us, aiming to obtain a reconstructed input that is comprehensible to us as humans too, and that can give hints on how the model interprets the input we give it. This approach remained largely unexplored until now, as we seen a lot of explainability methods, that were focusing on other aspects, though. The process of transforming embedding into audios, from now on, will be identified as *sonifying* and the *sonifications* are the results of that. Only by this genre of approaches it is possible to interpret the information it uses to make its decisions and predictions. In order to achieve this, me and the professors developed a novel approach that uses a gating layer as a bridge. This is used to connect two larger models and is done in the embedding/latent space, the internal hidden representation of these models that remained largely unexplored.

The whole procedure will be explained in detail in Section 3, while experiments on the different strategies are shown in Section 4; obtained audio results are available at this link; code available by clicking here.

Chapter 2

Related works

2.1 Audio Domain Topology

Here I want to focus a bit on the audio domain, which is divided into three macro-areas characterized by the type of sound that is processed: speech, audio, or music. Note that these areas are not strictly divided and the borders from each other are blurry.

Speech processing includes audio in which one or more people are talking. In this context, transformers can be used for tasks like:

- Emotion recognition: analyzing emotions through lexical and/or paralinguistic features (cadence, laughter, volume, etc.);
- Speech to text: transcribing from audio speech to text;
- Language detection: detecting the language spoken by the interlocutor in a given audio;
- Speaker/Voice recognition: predict who is talking among a series of speakers;
- Speech translation: translating audio speech into a language different from the original.

Audio events, on the other hand, involve the waveform analysis of the most various events, where we might encounter background noise, conversations, animal sounds, vacuums, engines, etc. Active research areas in this domain include:

- Audio signal analysis: extracting useful information from the audio sample in order to classify or synthesize them;
- Sound classification: assigning a label to an event sound;

- Audio enhancement: improving the quality of audio or reducing background noise;
- Audio source separation: separating elements that make up the audio, for example, if multiple people are talking at once;
- Spatial audio processing: analyzing and processing audio focusing on its spatial features.

Finally, the **music processing** domain involves the analysis of audio samples that define music, from complete songs to individual digital and/or electronic musical instruments. Research tasks in this field include the following.

- Music genre classification: extracting the genre(s) of a song;
- Music information retrieval: obtaining information about the metrics, artists, or instruments involved;

2.2 Audio Transformer models

Audio transformer models have the same general architectures of the models depicted in Section 1.1 with a particular focus on the input/output data type.

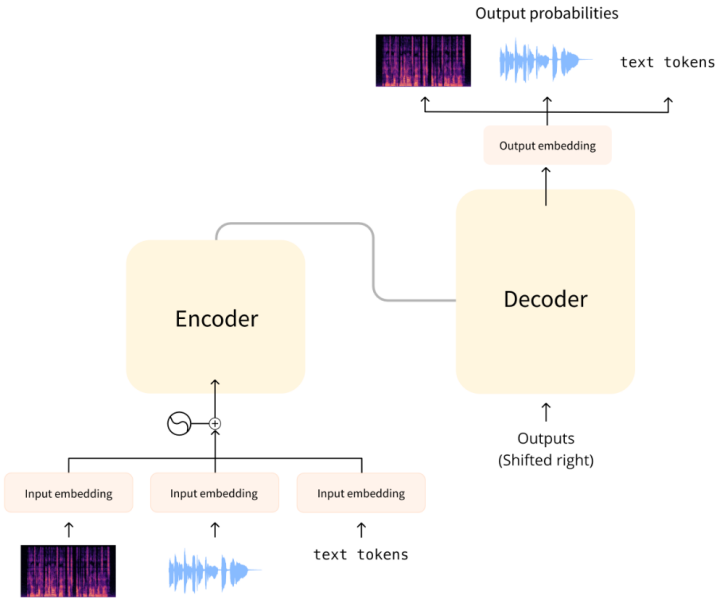

Figure 2.1: Audio transformers architecture; Source: *Hugging Face: Transformer architectures for audio*

According to the task they are conceived, the architecture could be different, as also focused in Section 1.1. If we think to classification tasks, audio tagging and similars these models use only the encoder part of the transformer architecture, while for generative task the decoder module is used. In order to see how audio transformers are used for generative tasks, refer to Section 2.3. As we can see, they process audio data and related data types, such as spectrogram images or text tokens. Each input type has its own features.

- **Audio waveforms** are listenable audio samples belonging to one of the categories of Section 2.1. They are usually processed from specific libraries; Python examples of these are *librosa*, *soundfile* or even *NumPy* that transform the audio waveform into one-dimensional arrays. Moreover, these libraries allow us to resample an audio waveform making them really useful especially in cases when the transformer model expects audio sampled at a specific frequency, as it was in mine. Before being processed, the waveforms are usually normalized. In this category, we find models like Wav2Vec2 by Baevski et al. [7] that masks the input before projecting it in its latent space, HuBERT by Hsu et al. [8] that overcomes BERT [2] architecture on which is based by using a masking process for its input and WavLM by Chen et al. [9] that focused not only on speech recognition, but also on several other tasks.

- **Audio spectrograms** are audio samples viewed in the frequency domain through an image. Usually on the horizontal axis we have the time dimension represented linearly, while the different frequencies that compose the audio are set on the vertical one and are represented linearly or logarithmically (useful for audios composed by a wide range of frequencies). The brightness of these stripes tells us how much amplitude has a certain frequency; so we expect that darker parts of the image are almost silent, while the brightest parts have a higher volume. We can note these aspects from the input/output image depicted in Figure 2.1. Since the spectrograms are fundamentally images they found large employment trough the different technologies. In fact, they have the advantage of being processed not only by audio models, but also from image ones. In this category, we can find models such as Whisper [10], AST [4] and its descending ASiT [11] and SSAST [12], ACT [13] that performs Speech-to-Text and SAT [14] that aims to optimize existing transformer-based models towards a streaming inference.

- **Text**. Most famous AI applications receives mainly text as input and outputs the data that we required. Note that if the transformer receives text as input and outputs text, it is not considered an Audio Transformer obviously. Since these are audio transformer generative models they will be covered in the upcoming section.

Despite the input type on which they are based, transformer models will always transform them into an embedding representation, that is the only way the transformer can process inputs; then the functioning is the same of the architectures shown before. AST and HuBERT will also be described deeply in Section 2.5 as they were used for this thesis research purposes.

2.3 Sonification models

Before proceeding with the thesis background, let us have a brief overview on some of the existing sonification models; these models are characterized by producing audio waveforms as output. In this field, we have some different model families that we can analyze. Please note that these model typologies can also perform outside the sonification task such as in image or time-series generation. However, since my thesis aims to the embedding sonification, I will analyze the general aspects of them by giving some insights on how these architectures are used in the audio domain.

Variational Auto Encoders (VAE) by Kingma and Welling [15] is formed by an Encoder model that compress an input to a latent space probability distribution by using a downsampling network, and a Decoder model that tries to reconstruct the value in the latent space into an output that should be as similar as possible to the input. We have to note that the latent space is represented as a continuous probability space; this differentiates the VAE from the simpler Auto-Encoders that use a discrete latent representation allowing the VAE model to obtain outputs that are similar to the original sample. Often, these models adopt the reparametrization trick and the KL-Divergence as loss.

Generative Adversarial Networks (GAN) were introduced by Goodfellow et al. [16] and are based on adversarial training of two different neural networks: the generator and the discriminator. The generator aims to generate new data based on a certain probability function, while discriminator aims to distinguish if a received image is real or generated by the generator network. The parameter optimization of the two networks is done with respect to the loss computed by the discriminator after the comparison. The model weights are optimized alternatively: the generator parameters are optimized by keeping the discriminator parameters freezed and vice-versa. Within this family we can find HiFi-GAN by Jungjil K. et al. [17] that achieves model performance and high precision in audio reconstructing by exploiting periodic patterns during sound generation. Some versions start from a text prompt that is given to a tokenizer. This outputs a spectrogram that is processed by the GAN that generates the output audio waveform. Descript Audio VAE by Kumar et al. [18] is another approach that will be explained deeply in Section **??** as it was part of the project background.

Diffusion probabilistic models or, simply, diffusion models were introduced by Sohl-Dickstein et al. [19]. They are class dependent latent generative models and the main interesting idea comes from the physical non-equilibrium state. If we put a colored drop in water, we can clearly see where the stain is. If we shake the water container the color mixes with water until the equilibrium is reached and we have colored water. With this idea Sohl-Dickstein et al. thought about a model that, during the first phase, could learn to bring complex distribution to more tractable ones through the forward or diffusion process. The generation procedure, instead, tries to reverse this path. We can see an exhausting survey on these models in the article published by Yang et al. [20].

Since every model works within a latent space where complex data are transformed in simpler forms, me and my supervisors propose an approach based on the gating layer model. This maps the embedding data sample representation of a certain model into the latent space of another one belonging to the same input domain acting as a sort of bridge. In this way, the focus can be put also on the hidden representation of the models that has not been covered yet in existing literatureas shwon in the next section regarding transformers related explainability methods.

2.4 Transformer explainability

How transformers interpret input data remains largely opaque. For this reason, it is necessary to adopt explainability techniques, in order to make them trustworthy and accountable for their predictions. Despite Explainable AI is an almost new research field, some transformer explanation techniques already exists. An important one is the Layer-wise Relevance Propagation (LRP) [21] that operates by backpropagating the prediction relevance using ad-hoc developed layers. It works well addressing the most influential parts of the input that are related to a certain prediction; they could be then highlighted with a colormap becoming very interpretable. Other works start from this method and try to expand it. This is the case of Chefer et al. in [22] where they provide the relevances information for each attention head that was not so well addressed in LRP.

Another way is the raw attention exam that could be helpful by providing hints on how the information flows among the model layers and consists in observing the attention matrices, often converted in images, used by the model to convey useful insights. Perikos et al. [23] use this method to infer sentiment analysis starting from text input and they found how much focused was the model on several input words. Another interesting approach was AttentionViz visualization toolkit by Yeh et al. [24], but it only works with language and image processing models.

A brand new approach was built by Akman and Schuller in their AttHear

[25] that is one of the few attempts of sonification in the audio transformer model explainability field. Their method uses a novel formulation of Non-negative Matrix Factorization (NMF) combining also attention weights and time activation information. In this way, they retrieve useful input sample parts that have major information and make them audible to the final user. Despite that, the embedding representation used by audio transformers remains unexplained.

An overall survey, wrote by Fantozzi et al. [26], depicts the general status of transformers interpretability analysis updated to 2024. They organize the transformer explainability into 5 macro-categories that depend on which transformer feature is exploited. Those are: Activation methods (like LRP), Attention methods, Gradient methods, Perturbation methods and the Hybrid one that combine the four previous ones two or more at a time. After seeing their work, we can say that the number of interpretability methods for this model type is rising.

Despite that, no one seems to have focused on the embedding representation, that remains largely unexplored. Until now seemed that all works addressed how the attention is remarkable in order to make a prediction or on which parts the model focuses the most, but not on how the internal model representation could be interpreted. That is why I wanted to focus on this aspect trying to achieve a sonified version of the hidden state representation of these models. Through the gating layer (see section 3), the transformer embeddings of a model that we want to interpret become audible thank to the mapping into the latent dimension of a sonification model. Now, to comprehend better the section that will come next, let us dive deep into the background architectures that I used: AST and HuBERT models and the Descript Audio VAE.

2.5 Project background

2.5.1 Audio Spectrogram Transformer

Audio Spectrogram Transformer, shortened in AST, is a transformer model built by Yuan G. et al. [4]. It was developed as convolution free encoder-only transformer. It receives as input raw audio signals of any duration, sampled at a frequency of 16.000 Hz; first, audio is processed by the model feature extractor. This module is responsible for transforming the audio encoded as a linear vector into a filter bank of a fixed shape of [1024, 128]. The first dimension depends on the audio duration, while the second is related to the quantized audio frequencies. This is basically a padded spectrogram, as we will see in Figure 3.4. This matrix is then normalized by subtracting the column means and padded or truncated to the fixed dimension above forming what will be the transformer model input.

Inspecting model code, the first step projects the filter bank into the embedding space through the patch embedder that transforms it into a 3D matrix of shape

[1, 1212, 768]. Then, the model embedder adds to this matrix two tokens (called classification and distillation tokens) that are used for classification task. The embeddings, that, at this point, we can call first hidden state, are forwarded through the 12 model layers. At each layer, 12 attention heads compute the attention score and update the hidden state value. At the end, an ASTMLPHead layer computes a linear projection and the class prediction by computing the mean of the token of which we spoke before.

2.5.2 HuBERT

HuBERT (Hidden unit BERT) model was developed by Hsu et al. [8] where they adapt BERT model to the audio domain. Furthermore, HuBERT exploits BERT masked prediction strategy, but its aim is to predict audio related features instead of text tokens. HuBERT masking strategy is derived from Wav2Vec2 [7] and the loss calculation is a weighted sum of two cross-entropies: one computed on the unmasked steps and the other on the masked ones. They also use different k-means models to improve the quality and the granularity of codebooks extraction. These cluster ensembles are being refined during the training process in order to start to well-known representations and find also new ones. It was developed into three versions: base, large and extra-large. For this thesis research purposes HuBERT base pre-trained on 960h librispeech was employed since, differently from AST, it processes directly the audio waveform, without passing from audio Mel-Spectrogram representation.

AST and HuBERT models were used as test model from which to extract embeddings to perform gating layer mapping test towards the next model latent space.

2.5.3 Descript Audio VAE

Descript Audio VAE was built by innnky (here is its github reference [27], that modified RVQGAN by Kumar et al. [18]. The modifications consist of replacing the RVQ module with a VAE, which is easier to train on for some models. This method was useful during the final part of sonification phase. As sonification is intended the process of an embedding representation into a reproducible audio waveform. By listening to this audio we can analyze the reasoning process of the transformer model by the perspective of its internal input representation.

RVQGAN allowed them to compress audios 90 times more than the state of the art and obtaining reconstructed audio that are very similar to the original one. They achieved this through an interesting research process, active on different aspects. First of all they put Residual Vector Quantization to the already existing VQ-GAN embeddings improving its scarce use of codebooks. They also used factorization

codes, allowing code lookup and code embedding in two different dimensional space, reducing the lookup dimension by maintaining the same code embedding size, that is directly related to the information that they carry. L2-normalization helps stabilizing the model with these codebooks. Furthermore, they used the MultiScale Discriminator and Multi-Period waveform discriminator strategy inside the decoder logic, adding an STFT discriminator to improve the audio phase modeling. Finally, they trained the model by using a different loss types such as L1 loss computed on spectrograms of different window lenghts; they applied a codebook loss as it was in VQ-GAN and then merged all these losses together by a weighted computation.

Chapter 3

Methodology

Gating layer model was employed to map the AST/HuBERT hidden states to the Descript Audio VAE latent representation. The implementation could be of any type: from linear, to convolution by passing also from the attention.

The gating layer was trained to map the embeddings deriving for the AST model to the latent representation of the Descript Audio VAE. My goal here was to achieve good sonification results starting from the embedded representation of an audio provided by a certain model. This procedure was then repeated starting from HuBERT embeddings. The key here was trying to map the embedding representation of a model that processes raw audio waveforms instead of the associated spectrogram representation in order to observe if the proposed model could fit well with that approach. The main pipeline is in Figure 3.1.

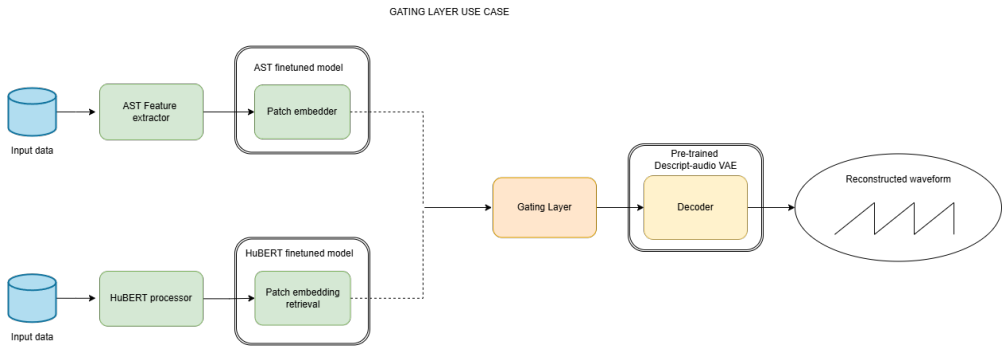

Figure 3.1: Gating Layer methodology workflow. From the left: Green boxes are the model we want to interpret truncated after its embedding computation. Dotted arrows mean that only one model at a time stands in the pipeline (gating layer is trained on specific model embeddings). The orange box is the gating layer, while the yellow one is the sonification model decoder that transforms latent representation provided by the gating layer into a reconstructed audio.

For the deep explanation of the models around the gating layer refer to Section 2.5.

3.0.1 Dataset

Gating layer dataset required AST/HuBERT embedding representation and the latent representation as ground truth in order to compare what the model produced with a real latent representation. To achieve this, all the audio samples of ESC50 [28], the chosen dataset, were processed with models said before.

ESC50 is a labeled collection of 2000 audio samples coming from environmental sounds. As the name suggests, there are 50 samples classes and each of them has 40 samples, producing a balanced dataset. Each sample lasts 5 seconds and it is sampled at 44.100 Hz. Three partitions were made from that: training (80% of the dataset), validation (10% of the dataset) and testing (10% of the dataset). Filenames belonging to the different partitions were noted on text files in order to have a deterministic starting point for each training session.

In order to compute the **embedding representation** audios were resampled to 16 kHz since it was necessary for both AST and HuBERT models; in fact, they expect audio sampled at that frequency. This was made with *librosa* library.

AST required importing its feature extractor and the previously finetuned model on the same dataset. This was not imported from Hugging Face spaces or from pre-trained models in general to respect the dataset split spoken above. After that, the patch embedder module was extracted from the finetuned model since the focus was on computing audios patch embeddings instead of doing the whole classification procedure. Each of the resampled audio samples passed through the feature extractor that computed the audio filter bank (ref. Figure **??**). AST embddings of ESC50 samples were obtained by passing feature banks through the patch embedder module. Finally, they were saved *torch*.

HuBERT, on the other hand, has not a patch embedder model. So an editing of the source code was necessary in order to have a portion of the original pipeline on which we were interested. Embeddings were extracted before the masking step, right before passing them into the HuBERT encoder.

Latents representation were computed from Descript Audio VAE model. It is composed from an encoder and a decoder modules; it uses the encoder to obtain the latent representation from an audio; this time the audios the model wants are sampled at 44.1 kHz, so no resampling was necessary in this case and then it saves it at a specified path. By doing that, a suitable dataset for the gating layer model was built.

In Figures 3.2 and 3.3 we can see an AST embedding and a latent Descript Audio VAE representation images. HuBERT embedding is not reported, but it is similar of the AST one, just with a different width and height. As we can observe,

these representations are totally different from a spectrogram, and they lack of interpretability. So, if we could obtain a significant representation from those, we could find other methods on how information flows in transformer models.

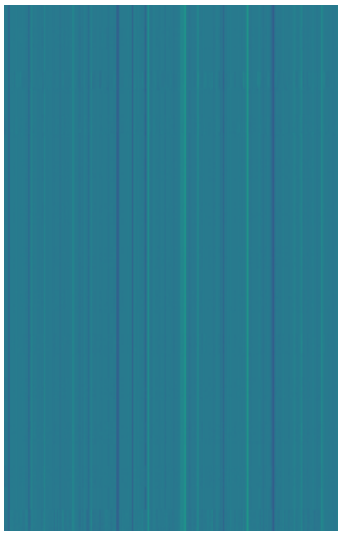

Figure 3.2: AST model embedding for sample 1-100210-A-36

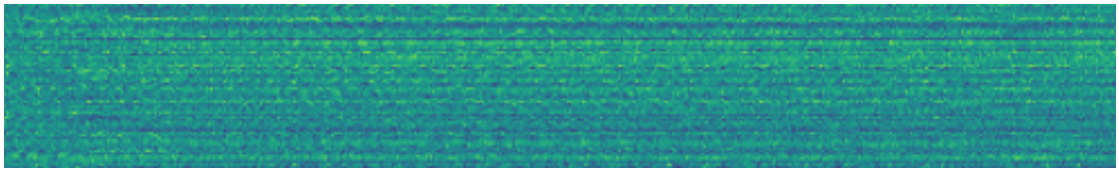

Figure 3.3: Descript Audio VAE model latent for sample 1-100210-A-36

3.0.2 Temporal dimension alignment

An important reasoning has to be focused. Since the shape of the final results must be equal to the latent representation, at the end of the computation process done by the gating layer model, a reshape was simply applied to the processed embedding. This was made without thinking on which of the two dimensions was the temporal one, both speaking in term of embeddings and latents representations.

This observation is mandatory since if we map a certain matrix into another one we must be sure of mapping the dimensions accordingly, otherwise results may be affected and, in fact, this lead to better sonification results. Basically, every encoded audio has a certain shape, usually [1, n_samples] where n_samples depend on its duration and the sampling rate used to encode the audio. This encoding has usually the temporal dimension flowing from left to right, so on the horizontal axis.

In the embeddings and the latent computation, this audio is processed by several transformations, so where is the temporal dimension oriented after that? Do they change it?

To address these questions, an analysis for the embedding computation and another for the latent representation were made. Regarding embedding representations, their computation involved the feature extractor and the AST model patch embedder. The feature extractor output is the filter bank cited in Section 2.5.1 that, if observed through an image, is just a padded (or truncated) vertical spectrogram where the temporal dimension flows bottom up.

N.B: *For the sake of space occupation, Figure 3.4 is rotated at a 90-degree angle. Imagine it vertically oriented with the padded part (the flat green one) standing at its top.*

Figure 3.4: AST filter bank of shape [1024, 128] rotated by 90 degree obtained by processing an audio file with AST feature extractor.

Since the filter bank is a spectrogram representation, it could be transformed into sound. So, it makes sense to consider its temporal dimension as a starting point. Now, this padded spectrogram passes through AST patch embedder that transforms this matrix into another one of a fixed shape that the model can comprehend: the embedding representation. In doing so, the filter bank is submitted to some operations that have to be analyzed in order:

1. Unsqueezing: this operation just adds a single dimension along the specified axis, so it does not change the temporal dimension orientation.

2. Transposition: this operation rotates and flips the input matrix, so, after this, the temporal dimension flows from left to right.

3. 2D-Convolution: this operation simply computes the convolution between the input matrix and the layer weights and bias, so no dimension orientation changes. The output of this operation has three dimensions (that are: [768, 12, 101])

4. Flattening: this operation converts a 2-dimensional matrix into a vector. This layer is applied to the last two dimensions of the output of the previous convolution; thus we obtain a new [768, 1212] matrix, which still has a horizontal flow of the temporal dimensions.

5. Transposition: as said above, this layer rotates and flips the temporal dimension, now making it flow vertically and obtaining a final shape of [1212, 768].

So, AST embedding representations have a **vertical temporal dimension alignment**. The above reasoning was made also for the HuBERT models leading to the same result. Those transformers models have embeddings that are of a different shape, but the time dimension orientation is the same.

Let us observe latent representation related to Descript Audio VAE. Here, the flat audio (shape: [1, n_samples], again, horizontal temporal dimension) follows a more complex treatment. We can refer to it from the list below:

1. Weighted norm 1D-Convolution: this is a normal 1-dimensional convolution embedded into a weight normalization layer. Both the normalization and the convolution, as we said before, do not change the temporal orientation.

2. 4x Encoder blocks: please refer to Appendix A to see the encoder block expansion, but here we can say that after this the temporal dimension orientation does not change, staying horizontally oriented.

3. Snake1d activation: since the activation functions maintain input values in a certain range, this is similar to a clipping, so no changes have been made on the axes.

4. Weighted norm 1D-Convolution: as said in the first point, no changes on the temporal dimension.

This procedure is repeated few times by changing the kernel size of the convolutional layers passing from different matrices shapes. The end point is reached for the final latent shape [64, 431]. Since this procedure does not change the temporal dimension orientation for the first time, it does not change it if iterated. So, the temporal dimension remains unchanged from the beginning to the end, so it is **oriented horizontally**.

This analysis highlighted an important fact: embeddings have to be transposed before being mapped to the latent representation in order to maintain the temporal dimension correctly aligned. Furthermore, these results can also be seen by the lines orientation in Figures 3.2 and 3.3.

Another, final, observation on the embedding dimension has to be made. AST embeddings discussed until now are considered the ones that were output of the patch embedder and not from the embedder layer, that is the actual AST hidden state representation that is shaped [1214, 768]. This was done because the embedder outputs the patch embedding output after a head or tail addition of two tokens (each one of shape [1, 768], called classification and distillation tokens respectively)

and that are responsible of the classification process. So, in order to give the gating layer the most pure data, avoiding any possible bias, the input of its computation is set to the output of the patch embedder that has the shape of [1212, 768] and does not have any information about a possible belonging class of the current sample.

3.0.3 Training

The training phase was done several times in order to train different model versions, but it always been almost the same. In this section I will describe the general workflow.

The first step was defining the model parameters that impacted on the training process:

- Training batch size=64;
- Evaluation batch size=32;
- Logging steps=2;
- Training gradient accumulation steps=2;
- Evaluation gradient accumulation steps=1;
- Seed=42;
- Metric for the best model: loss;
- Logging strategy: steps;
- Save strategy: epoch;
- Evaluation strategy: epoch;

These values were the same among the different computations. Some others were kept to the default ones that the Trainer object already had.

After setting those values, it comes the model training.For each model version, different learning rates were tried. Initial values were on a large scale e.g. [1, 5e-1, 1e-1, 5e-2, 1e-2, ... 1e-5, 1e-6] combined with a reduced number of training epochs (15). The model training, as anticipated, was made using the Hugging Face Trainer and TrainingArguments classes. These objects can be very useful since they already implement the forward pass, the gradient computation and the backward step that must be executed during the model training phase. The models training were made on the training dataset partitions described in Section 3.0.1. Since it was a mapping task, loss measure was used as evaluation metric for the training process: the lower the loss, the better the model. This was calculated by comparing the

distance from the gating layer generated latent to the ground truth one in term of Mean Squared Error (MSE) loss as this type seemed suitable for the task I was trying to solve. During the training procedure, Trainer object also saved the model checkpoints after each epoch that are useful to continue the training from a certain point.

After training, results analysis was performed provided by the Trainer Hugging Face class as logs with *tensorboard* [29]. It is a useful toolkit that allows to analyze different model statistics in the training and evaluating phase such as the computation time, number of samples per second, learning rate variations and loss values. It has a smoothing option that regularize original loss curve that is usually very segmented. This option uses a value in the range [0, 1]: 0 returns the original curve, 0.99 the most smoothed one. This is helpful to focus on the trend of the loss curve that should decrease for lower epochs becoming way more constant as the epochs pass. I always used smoothing factor 0.99 unless differently specified. So, after analyzing logs statistics in term of training and evaluation loss curves learning rates and the model versions related to lower loss curves were selected. These were set as starting point for further experiments involving more specific learning rates that were trained for a higher number of training epochs (50).

3.0.4 Testing

After finding the most promising gating layers based on the loss metric, I tested them by doing inference using the test set: split that was left unseen for both AST/HuBERT finetuning and gating layer training. In first tests, this phase was done considering only one sample. Then, after different experiments on different model versions, data augmentation and data normalization pre-processing and different post-processing rescaling and filtering types, 77 samples were considered taking care of bring at least one sample for each of the 50 classes belonging to ESC50 dataset. Inference phase started with loading an already finetuned AST/HuBERT model and extracting its modules for embedding computation. Next, the dataset audio file was processed by these and its embedding representation was computed. Descript Audio VAE model di not required any finetuning process since it performed by inferring on an audio sample from ESC50 by using the whole encoder decoder model. The track was sonified in a good way so the finetuning was not necessary. After loading the Descript Audio VAE decoder, the steps made were:

1. audio loading after a 16 kHz resampling;
2. AST/HuBERT embedding extraction from the audio;
3. embedding mapping into the latent representation of the Descript Audio VAE through the trained gating layer model;

4. decoding the latent representation returned by the gating layer with the Descript Audio VAE decoder model obtaining an audio waveform.

This procedure gave me good and bad results that will be discussed in the following section. Moreover, all the results are available at link shown in Section 1.3. Despite that, the waveform amplitude was lower with respect to the original one, which had a larger waveform shape on the vertical axis. So, sonification values range was checked w.r.t. the original audio. The range of the original audio was almost between [-1, 1], while the sonified one was in the range [-0.4, 0.4]. So, the audio has been rescaled as I will show in Chapter 4.4. First results were treated as tests and were tested with a *subjective analysis*. Most promising results, later, were compared in this way and, furthermore, through an *objective analysis*.

This last one was about to compute the FAD and KAD metrics on the obtained results. These metrics are useful to compare audios between them and obtain a numeric result from which we can derive objective thoughts.

FAD stands for Fréchet Audio Distance and was introduced in 2019 by Kilgour et al. [30]; they noted that existing audio evaluation methods often disagreed with subjective evaluations made by human listeners. Their aim was to introduce a new audio evaluation method that had to compare how a generated sample with a clean studio recorded one. The main procedure consists on collecting two large sets that are called target set, related to the generated audio samples, and the reference set, composed by studio recorded samples, the ground truth of our experiments. These are compared together after an audio embedding extraction made using a pre-trained classification audio model; next, Gaussian mean and covariance are computed from the audio embeddings and the Fréchet distance is computed on the distance of the two multivariate Gaussian estimated on the embeddings. This comparison method has been used in literature until these days, but it has to deal with two main issues. First of all, it is set size dependent; for the reference and target set MVGs are extracted, so these sets should have enough samples to model those distributions. Then it assumes that audios follow a Gaussian probability density function that is a good approximation, but not always true on generated samples.

KAD, recently introduced by Chung et al. [31], overtakes these issues as it is not set size dependent and the data distribution it assumes is not Gaussian constrained. Its formulation derives from a weighted Maximum Mean Discrepancy (MMD): a statistical test used to distinguish whether two samples belong to the same distribution or not. This is done by evaluating sample relationships through an unbiased Gaussian RBF (Radial Basis Function) kernel. For these reasons, I computed FAD measure to be consistent on what was done in previous research projects, but I also computed KAD metric as it should be better formalizing the audio comparison problem. This was done by using the KAD toolkit provided by

Chung et al. [31] and computing the comparison between the different sonifications files against the original one.

Qualitative analysis, on the other hand, was made comparing sonified files among them and the original dataset sample through a free software for audio editing called "Audacity" (see `https://www.audacityteam.org/`). Test results were evaluated by viewing how much their waveform was similar to the original sample and also by listening to them. Finally, deeper qualitative analysis was made on the 77 samples cited before. The aim here was to provide a human thought and see if the results were aligned to what the objective analysis returned. Sonification samples from the top 5 different versions of the gating layer were chosen.For those, additionally to the standard qualitative analysis explained above, a mark from 1 to 5 (1=very low quality/totally different from the original, 5=high quality) was assigned using as metric the audio similarity and the waveform shape similarity with the original audio. Next, marks means for each model version were computed. From them it has been possible to have a subjective metric for the process made by each of the chose gating layer versions. More details on these results will be given in the upcoming section.

Chapter 4

Experiments and results

In this section we are going to see the detailed process of the model building by discussing and analyzing the different performed experiments.

HPC by Politecnico di Torino was used to compute embeddings and latents representation computation, the training and the inference phases. I used Legion Cluster that consists in 24x nVidia Tesla V100 SXM2 - 32 GB - 5120 cuda cores as GPU and 2x Intel Xeon Scalable Processors Gold 6130 2.10 GHz 16 cores as CPU.

4.1 Initial models testing: LNR, CONV, ATTN

As said in Section 3 about the different strategies the model should have followed, three initial versions were implemented.

1. **v0 - LNR**: A fully linear model that simply has two linear layers that converts the embedding size into the latent one.

2. **v1 - CONV**: Since the convolutional layers obtained good results in the majority of AI models a version with them should have been useful. In order to not overload the memory usage the kernel was set to be a (10, 10) square matrix and the stride to be [2, 1] rectangular matrix. They are followed by a ReLU activation function that prevents the effect of exploding or vanishing gradient. Finally, two linear layers maps the dimension derived by the convolutions into the latent dimension.

3. **v2 - ATTN**: Attention based model. This was copied from the attention mechanism of AST model and was used as gating layer. The number of attention heads was modified to 3. The rest of the values and the method was kept equal as in the original implementation. The attention computation is the same as shown in Figure 1.2.

Each of the model version cited before is a class extending *torch.nn.Module* and has the name of *"Layer_vX"* where X is the version number. The *gating layer* is also a class extending *torch.nn.Module* and is composed of a MSE loss function and one of the *"Layer_vX"*.

Most promising loss curves after a 15 epochs training are displayed in Figure 4.1. Those characterized by definitely too high loss values were excluded. These were obtained for higher or extremely low learning rates so those models were discarded. Discarding useless loss curves allows us to see clearly the trend of the best ones. Among these, we are interested on those models that have the best curves, but what is the rule to assess that? The answer is not well defined, but there are some reasoning that are commonly followed.

1. Loss curves tends to become constant by increasing the number of epochs. This has to be kept in mind since we have to choose those curves that have the chance to become lower and not the ones that are clearly at their minimum for a small number of epochs.

2. Overfitted models curves are usually characterized by learning rates that are in a range that does not match with the model itself, so we can discard them to choose other values that are in a most promising set of values. These learning rates, despite being lower than others, output often higher values loss curves, so, they produce models that perform worse.

At the right of this image, and of the majority of the figures that will follow there are also loss curves related to the validation process. In normal condition they should mimic the same behaviour of the training losses. If not so, we should pay further attention to the output of these models.

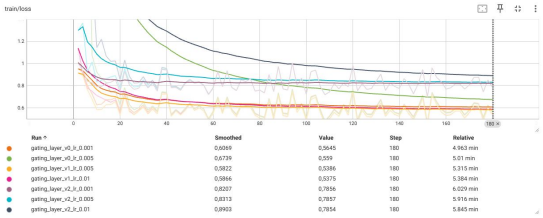

Figure 4.1: Gating layer LIN, CONV and ATTN best training losses after 15 epochs

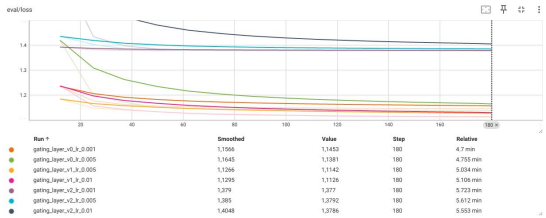

Figure 4.2: Gating layer LIN, CONV and ATTN best validation losses after 15 epochs

After a 15 epochs training only a small range of learning rates fits well for our model versions. This makes sense since the different version of the model are based on different architectures so each one has a set of learning rates on which

it performs better. We can see that CONV and ATTN, that are more complex solutions, have almost the same learning rates.

Now we can train those model versions for more epochs on a more specific set of learning rates that comprehend those on which the model performs better. Here the range of learning rates (often shortened as lr) [0.01, 0.005, 0.001, 0.0005, 0.0001] has ben used. Last two values, that were not present among the best ones, were also used to see if the models were really overfitting.

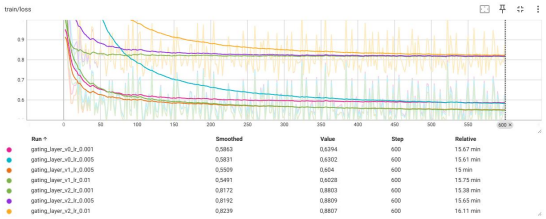

Figure 4.3: Gating layer LIN, CONV and ATTN best training losses after 50 epochs

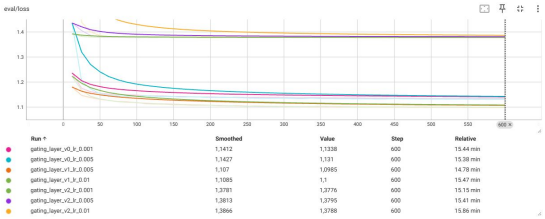

Figure 4.4: Gating layer LIN, CONV and ATTN best validation losses after 50 epochs

These results are filtered with respect to the most promising learning rates. The number of epochs turns out it is reasonable since the curves seems almost stable. We can conclude that the best models up to now are:

- LIN: lr=[0.005, 0.001]
- CONV: lr=[0.01, 0.005]
- ATTN: lr=[0.005, 0.001]

Finally, models were used to compute the final **inference** step. The followed approach was explained in detail in Section 3.0.4. Waveforms were obtained by mapping one casual test split embedding into the Descript Audio VAE latent space and then decoded by this model decoder. The result is presented in Figure 4.5.

Starting from the top we have the original audio sample named 1-7974-A-49, and is related to the usage of a saw and we can clearly distinguish the movement if we listen to it. If we can not do that just imagine that each peak of the waveform is a run of the saw. We then have a pause characterized by silence in between 2.5 and 3.5 seconds, then it resumes until the duration of 5 seconds where the audio ends. The sonifications are the following samples and, as we can see, are not so good, unless for the CONV model with learning rate=0.005 that is showing the most similar waveform among those compared. Other waveforms are just noise if referenced to the original audio. By listening to the samples we can confirm that

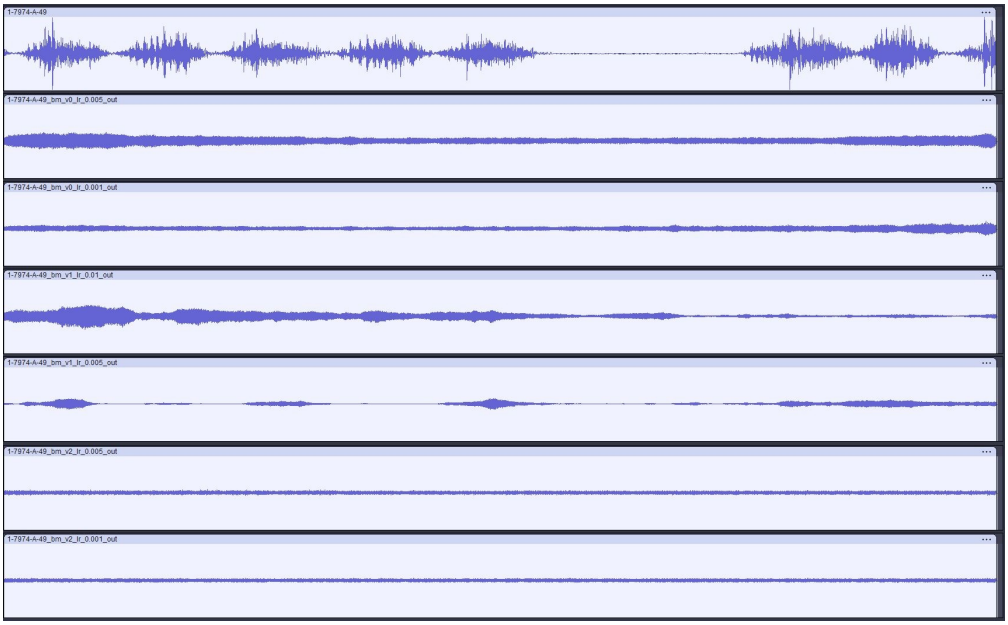

Figure 4.5: Sonification results for the different Gating Layer models after 50 epochs training

aspect. LIN version produced waveforms characterized by only continuous random noise. ATTN version audios are similar to those obtained with LIN, but are even most disturbed. The sound is similar to that produced by the electric current. Finally, CONV version has a strong similarity with LIN samples in terms of audio pitch. With respect similarity with the original sample CONV with lr=0.005 is the most similar, where we can also hear some changing pitch drafts related to the saw movement. Despite that, the audio reconstruction is not particularly good.

4.2 Temporal dimension alignment

Here I want to show a fast comparison between the model versions that **do not have** the input transposition, and those that have it taking back what was said in Section 3.0.2. I want to clarify that both procedures were tested on model versions shown in Section 4.1. The key difference is only the missing input transposition necessary to map the input temporal dimension accordingly to the output one. The results are showed w.r.t. model version. Before showing the sonification results, we can look at the training losses. For the sake of simplicity I will only report loss curves obtained at the end of the final training phase after 50 epochs.

By comparing them with the previous Figure 4.3, we can clearly see that the transposition of the input does not have such a big impact on the training results.

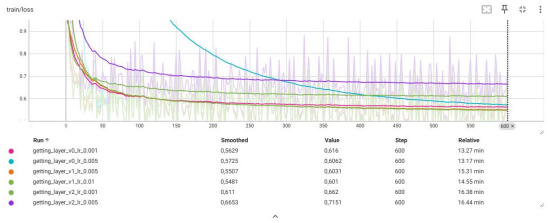

Figure 4.6: Gating layer LIN, CONV and ATTN with input transposition best training losses after 50 epochs

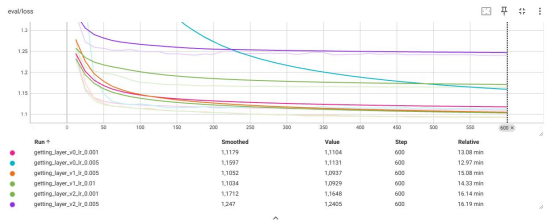

Figure 4.7: Gating layer LIN, CONV and ATTN with input transposition best validation losses after 50 epochs

Loss curves are quite similar in term of shape, depicting stable versions of the model; also the computation times are similar, aspect that should be the same as in this case since the model architecture was not changed. Despite that, within this shrewdness, loss values are lower; this should lead to better audio reconstruction than what we seen before, meaning that the temporal dimension idea was necessary. The model versions that have the higher loss drop are LIN and ATTN versions that were those that performed worse overall. Model version CONV, instead, has lower drop, but it is better than before too. Let us pass to practical results by seeing how inferred sonification looks like in Figures 4.9, 4.11, 4.13 by comparing them with the samples obtained with the gating layer versions trained without the temporal dimension artifact depicted in Figures 4.8, 4.10 and 4.12.

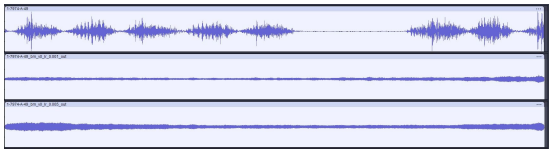

Figure 4.8: LIN 50 epochs: **no** input transposition

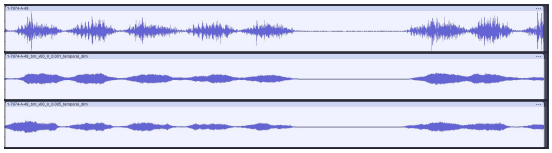

Figure 4.9: LIN 50 epochs: input transposition

From figures, we can say that sonifications obtained after transposing the input are definitely more precise than those obtained without except for gating gayer ATTN. We can clearly see how models LIN and CONV, trained with the input transposition, provide results that follow the input waveform behavior in a better way despite not being vertically high as the original sample. By listening to the sonified samples we can confirm these aspects noting that the pitch and the amplitude are similar among the sonified audios. By seeing that two versions over three are enhanced we can conclude that the attention based gating layer version is not so suitable for the AST embedding to the Descript Audio VAE latent space

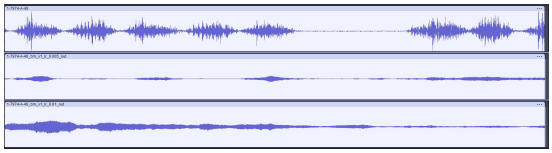

Figure 4.10: CONV 50 epochs: **no** input transposition

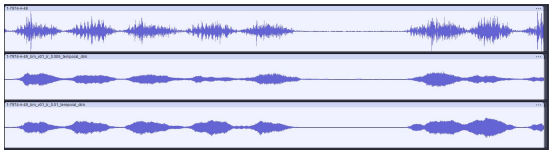

Figure 4.11: CONV 50 epochs: input transposition

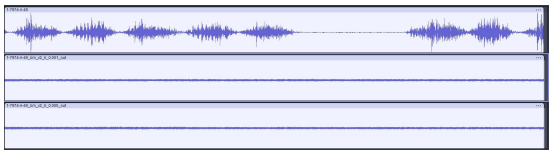

Figure 4.12: ATTN 50 epochs: **no** input transposition

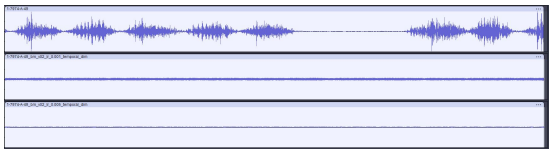

Figure 4.13: ATTN 50 epochs: input transposition

mapping specific case.

Since these model were retrieving good results and they had pretty different architectures they were taken as **baseline** for this thesis project.

4.3 Data augmentation & normalization

As a usual approach tried for improve model performances data augmentation and data normalization were also tested. There exist several methods to do both procedures.

Data augmentation was tried with *torchvision.transforms.v2* library since embeddings could be assumed as a sort of image with only one channel (as a gray scale image). This library offers different types of image processing techniques. The one tested was gaussian blur that consisted into an addition of a random noise to the embedding tensor that make its representation more opaque if we think at the embedding as an image. This transformation requires a kernel size and a range from where the sigma will be extracted. This sigma value would be the impact of the noise addition (the more it is high, the more the image would be noisy). After few tests made on a random image outside the embedding context, kernel was set to size=(7, 13), sigma=(0.1, 2) since the results obtained with these parameters were producing few noisy images among the whole batch, and those noisy images were affected with a decent noise quantity that made them blurred, but still understandable. This procedure was applied on the embeddings at the batch level extraction. In this way a blurred embedding would conduct to the same

latent representation seen during another epoch. Moreover, data augmentation was made only on the training set since the validation and test samples should not be affected by these type of transformations.

On the other hand, for the normalization procedure, min-max normalization formulation was used. It is shown in equation 4.1.

$$normalized\_output = (input - input_{min})/(input_{max} - input_{min}) \quad (4.1)$$

Gating layer baselines were trained with these data augmentation strategies alone and with their combination in the same manner explained in Section 3.0.3. The best loss curves for the three versions of the model are shown in Figures 4.14 and 4.15.

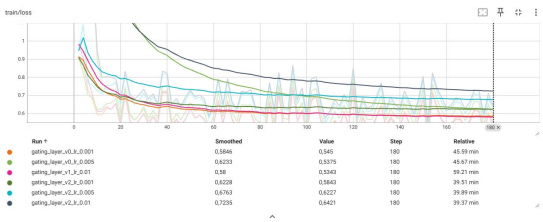

Figure 4.14: Gating layer baselines LIN, CONV and ATTN with blur data augmentation: best training losses after 15 epochs

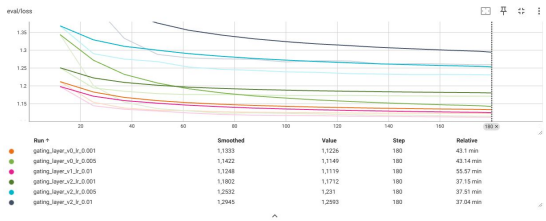

Figure 4.15: Gating layer baseline LIN, CONV and ATTN with blur data augmentation: best validation losses after 15 epochs

After 15 epochs training, data augmented model versions returned loss curves that were so similar to those obtained for the baseline indicating that the Gaussian blur effect is not very useful for the training process and does not affect so much also the evaluation. We have also to observe that computation times are more or less 10 times higher than the ones related to the baseline. For these reasons, gaussian blur data augmentation was discarded.

Now, Figure 4.16 and 4.17 show training losses of baseline models after applying embedding and latent normalization.

If we look at it overall we can see that loss values are the lowest up to now, so we may conclude that the best losses, and, consequently, that these are the best models; but, if we have a closer look to the non smoothed version of the curves we can find an interesting aspect.

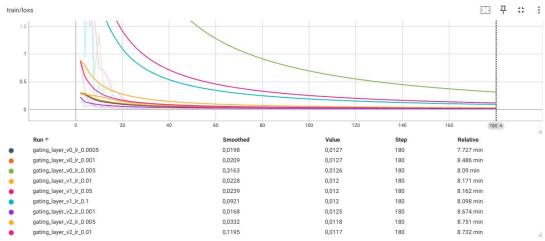

Figure 4.16: Gating layer baseline LIN, CONV and ATTN with embedding and latent data normalization: best training losses after 15 epochs

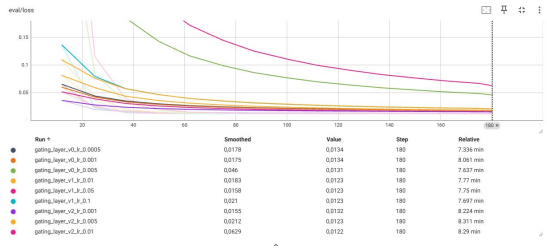

Figure 4.17: Gating layer baseline LIN, CONV and ATTN with embedding and latent data normalization: best validation losses after 15 epochs

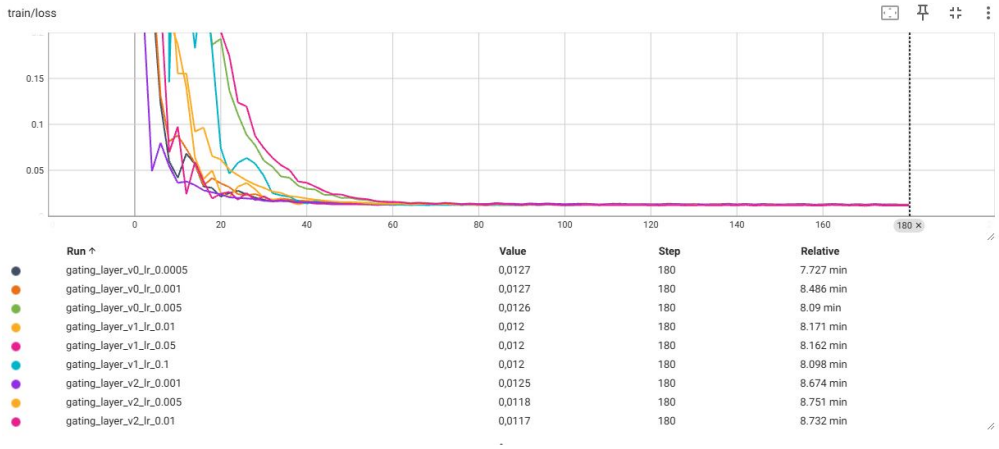

Figure 4.18: Gating layer baseline LIN, CONV and ATTN with embedding and latent data normalization: best training losses after 15 epochs, NO SMOOTHED

Figure 4.18 shows how the models are saturating after few computation steps. We can see that after 60 steps (equals to 2.4 epochs) the model is not learning anymore since the loss curve becomes stable. So, we expect that the sonification of these model versions would not lead to significant results. Furthermore, we expect that also training this variation of the model for more epochs will be useless. This could be due to the fact that all the embeddings on whose the model is trained on comes from the same starting model (AST or HuBERT). So they belong to the same range of values that is equal across them, so normalizing input is useless in this case.

Finally, the combination of these two approaches was also tested. This approach has the computational time of the gaussian blur variant and the low loss values of the normalization part; since both input enhancement methods were discarded,

also this one will end up this way.

4.4 Reconstructed audio rescaling

As we can see from the previous Figures like 4.9, 4.11 and 4.13, despite the good reconstruction, all the generated waveforms are limited in their vertical amplitude. The original input sample is always vertically wider. These differences result in sonifications that are characterized by a lower volume.

In order to solve this issue, a standardization procedure is mandatory. Be careful that this procedure is not part of the model, but it is a post-processing method to make the generated samples more similar to the original one. The rescaling strategy depends on two parameters: the range of values the output should assume and where to apply the rescaling.

Regarding output audio values these were chosen to be in range [-1, 1] as a usual normalized audio or in min-max range, where min and max are the minimum and maximum values of the input audio sample.

On the other hand, for the place where the rescaling should have been applied the steps were at reconstructed audio level or at embedding level (once extracted from the model we want to sonify).

Results obtained on the sample seen until now are shown in Figures 4.19 and 4.21. For these and the image that will follow three sonifications are shown characterized by: no rescaling, [-1, 1] rescaling and minmax rescaling respectively. Since the model results are just rescaled by maintaining the same waveform shape, in order to see the differences in a better way among the different type of rescaling strategy, only the results coming from the best models up to now $\text{LIN}_{lr=0.005}$ and $\text{CONV}_{lr=0.01}$ will be shown.

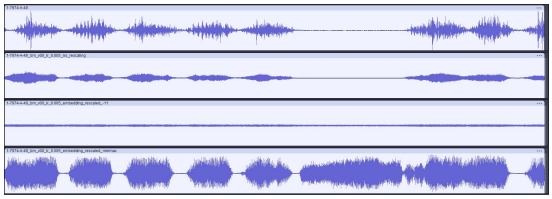

Figure 4.19: $\text{LIN}_{lr=0.005}$ sonification results after 50 epochs, embedding rescaling

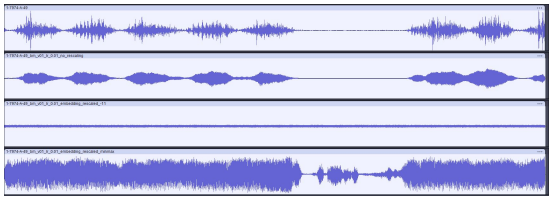

Figure 4.20: $\text{CONV}_{lr=0.01}$ sonification results after 50 epochs, embedding rescaling

In Figure 4.19 and 4.20 we have the sonification results for the model cited previously. We can say that the minmax rescaling was the best method for the embedding rescaling since the values belonging to that representation could be

in any range and if the rescaling is made in [-1, 1] a lot of information would be lost. Despite that, we can also observe that the embedding level rescaling changes dramatically the audios generated by both models that were following well the input audio shape. So, since things are getting worse, the embedding level rescaling method would be discarded.

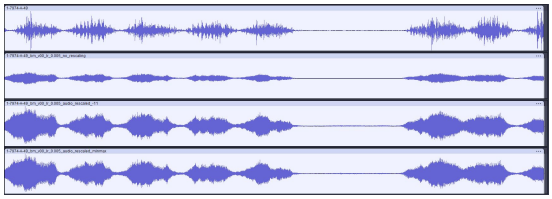

Figure 4.21: $\text{LIN}_{lr=0.005}$ sonification results after 50 epochs, audio rescaling

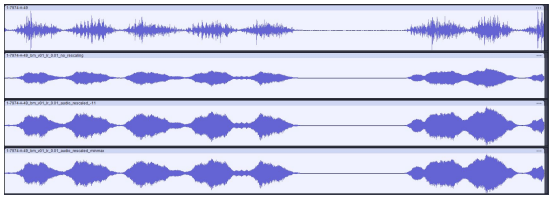

Figure 4.22: $\text{CONV}_{lr=0.01}$ sonification results after 50 epochs, audio rescaling

Passing to the reconstructed audio level rescaling shown in Figure 4.21 and 4.22 we can observe how efficient is rescaling on the final audio; the efficiency is given by a rescaled audio that still follow the original waveform shape, but has a wider vertical range, providing a sample that is more similar to the original one than the not rescaled version. The results for the ATTN gating layer version are not considered, as this model performed poorly as we can recall from Figure 4.13. Since this is a post processing method, not so computationally expensive, and that enhances output reconstruction, so it will be considered in future analysis.

Regarding the rescaling values range, from this audio, it seems that the two tested ranges are quite the same. This sample, in fact, is characterized by seven (and a half) well defined crests and a longer silence between the fifth and the sixth. Those crests are well distinguishable from the flat line. But what happens if we reconstruct another audio? ESC50 consists of several sounds and not all are characterized by these features. Let us have a look, for example, at sample 2-120587-A-6 that is labeled as *Hen* depicted in Figure 4.23 and 4.24.

This is the case of peaks related to a lower sound (lower difference between the crest and the silence). Audio rescaling in range [-1, 1] will produce rescaled audios that are characterized by a background noise since the rescaling strategy multiplies all the values inside the waveform by the maximum value of the input audio absolute value function, making the fine output belong to the whole [-1, 1] range. Overall, we can see that LIN overcomes CONV version in this sample reconstruction.

Furthermore, this aspect could be seen better by rescaling those sonified audios deriving from sounds like wind, opening doors, or flowing water that are characterized by very weak noises. An example is the *Pouring water* sample in Figure 4.25 and 4.26.

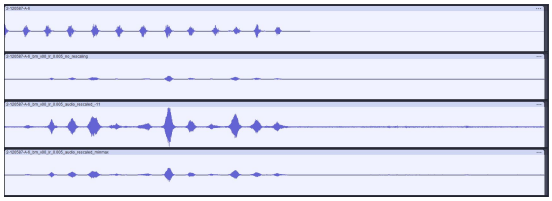

Figure 4.23: $\text{LIN}_{lr=0.005}$ 2-120587-A-6 sonification results after 50 epochs, audio rescaling

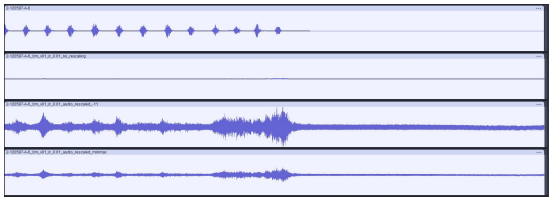

Figure 4.24: $\text{CONV}_{lr=0.01}$ 2-120587-A-6 sonification results after 50 epochs, audio rescaling

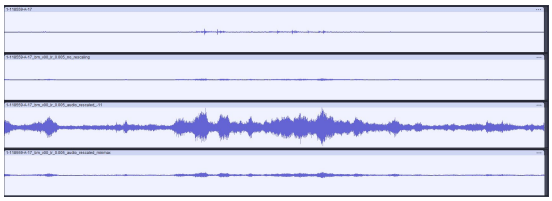

Figure 4.25: $\text{LIN}_{lr=0.005}$ 1-118559-A-17 sonification results after 50 epochs, audio rescaling

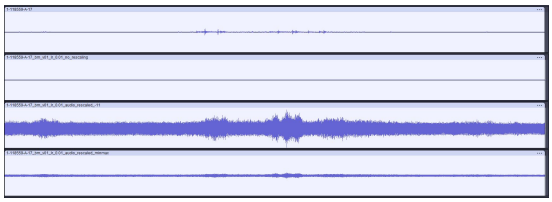

Figure 4.26: $\text{CONV}_{lr=0.01}$ 1-118559-A-17 sonification results after 50 epochs, audio rescaling

Another time minmax rescaling returns better results that the [-1, 1] range rescaling. For this reason, minmax rescaling was used as post-processing technique. In this way, the output is more similar to the input sample and the background noise is kept lower. Despite that, this rescaling method enhances background noise too, and this can be noted especially where there should be a silence. In those parts, each sample seen until now has at least a small background noise component that should be avoided. This is an intrinsic issue since the rescaling multiplies each audio value by the same quantity.

So, output audio filtering was focused too. In this way, we could apply an audio rescaling post processing on the whole dataset and, maybe, improve audio characteristics reducing background noise with the additional filter. So, few audio filters were also tested trying to remove the background noise. Note that these tests were made only on the audio cited before as they can be seen as critical audios, with very different characteristics among them. After these tests, on fewer samples, most promising filters were selected and applied to all the sonifications to research if it could leverage or not the output audio quality. These filters were applied after a minmax rescaling of the audio provided by the sonification model.

Different filtering approaches were observed, following diverse ideas for each of them. The main ideas were basically three:

1. **Threshold filtering**: every output audio values under this threshold is set to 0.
2. **Ad-hoc function filtering**: the output audio is processed by a function retrieving a filtered audio.
3. **Wiener filtering**: an existing filtering function.

First approach is thought to setting to 0 all the values below a certain threshold. Different tries with different thresholds values were experimented. Precisely, threshold was set to a certain percentage of the output audio, making this solution scalable across the different samples. The percentage of chosen thresholding were: 5%, 7%, 10%, 15% and 25%. The best one was dependent to the audio sample. This value was chosen related to the quantity of noise that has been filtered and the quality of the audio that was left. Lower threshold was not filtering enough, leaving the majority of background noise, while higher thresholds, filtered too much, leading to final audios that have also significant parts set to 0. Next, not all audios had the same waveform height as we seen; this aspect silenced some significant audio peaks or performed very disturbed samples. This lead to a method that is not general, but very specific to the sample we are processing.

In fact, this way of filtering will produce some sort of segmented audios: audio characterized by a segmented waveform instead of a continuous one. If the value in that instant is below the threshold that will be set to 0, independently to what there is before or after.

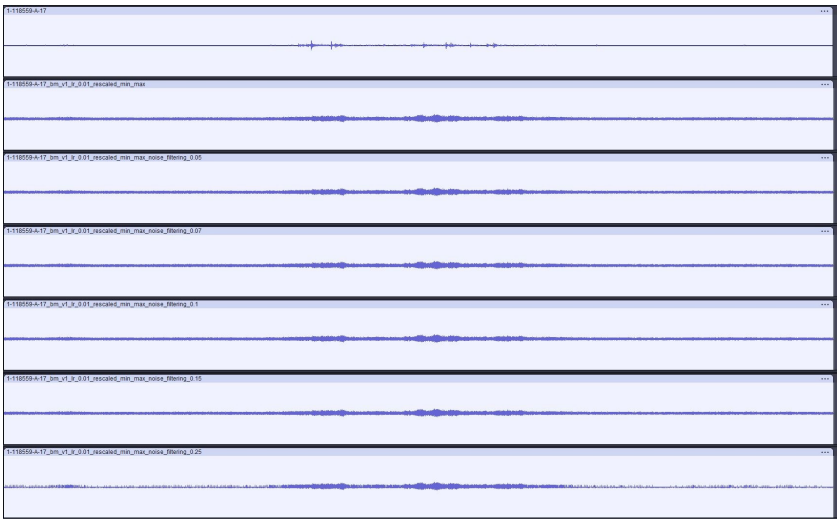

Figure 4.27: Example of segmented audio waveform.

An example of this phenomenon can be seen in Figure 4.27 where we have the original audio as first, the second one is the rescaled audio using the minmax

approach and then we have audios filtered using the different thresholds filtering. All the considered audios were produced by the same model. As we can see from the last sample, the critical part, where in the original sample there is silence, is filtered in a segmented way. Furthermore, in this particular case, the threshold should be augmented to better filter the background noise in these areas. Since the approach is very dependent from the processed track it was discarded.

Thinking about this aspect, I came up with the second approach that simply computes the exponential of the audio with a certain value. The exponents that I tested were 2 and $\sqrt{2}$. The idea was quite simple: to elevate each value of the output audio file to a certain number making the values below 1 lower, and those above higher, by maintaining a continuous representation of the audio file/waveform. This technique is also invariant to the waveform vertical amplitude, by working always the same for all the audios, independently the sound they are. Through the visual and auditive tests performed on these exponents 2 was chosen as the best one. Results related to these techniques are in Figure 4.28 where from the top to the bottom we can find: original sample, non filtered sonification, sonification filtered with sqrt exponent and sonification filtered with 2 as exponent.

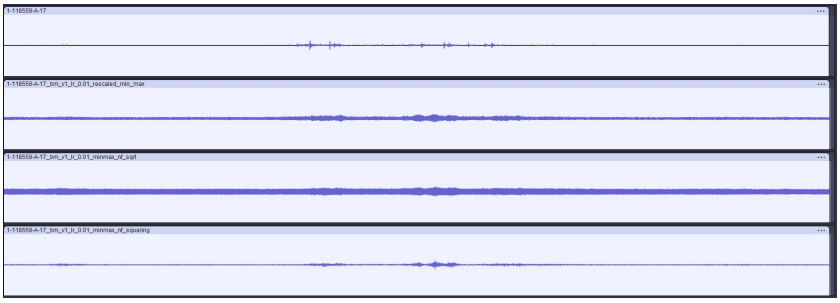

Figure 4.28: Example of exponential filtering.

Finally, also Wiener filter was tested; an already implemented approach available with the *scipy* library. It seemed that the filtered audio was almost the same (if not identical) to the rescaled model output (ref. Figure 4.29), this method was then discarded, in order to keep only the most promising one: the audio filtering with exponential 2 transformation that from now will be called *squared filtering*.

Concluding, after analyzing all the losses and the sonification results in a visual and a listenable manner (this was not covered in this document for obvious reasons, but you can listen to all the sonification by following the link at the end of Section 1.3), we can observe that the best gating layer models up to now are LIN with learning rate 0.005, 0.001 and CONV with learning rate 0.01, 0.005. These could perform even better if followed by a post-processing minmax audio rescaling and a minmax rescaling+squared filtering.

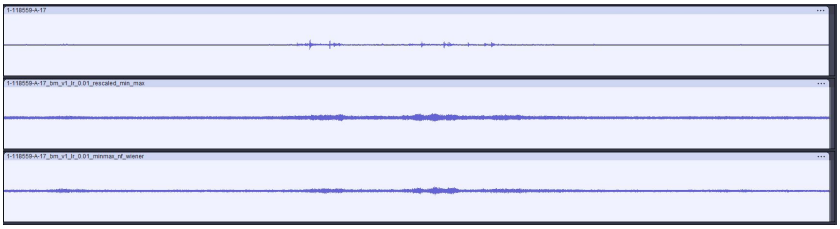

Figure 4.29: Example of Wiener filtering. From the top we have the original sample, the sonification rescaled into minmax range and the Wiener filtered sonification.

4.5 More model testing: LIN and CONV variations

After inspecting some rescaling and filtering techniques and their results, it is time to go back to the model architectures. We have completed our pipeline by having a few version of model architectures, some audio post-processing and we obtained good results from them also with rescaling/filtering combinations. We compared the results of the different techniques retrieving useful information on which were the most promising approaches. Now it is time to modify the existing architecture of the gating layer models trying different layer types in order to see how they affect the results and assess if they can leverage the model performances. Most promising versions, LIN and CONV, were taken and some layers were added to them, in order to obtain other versions that were related somehow. Different layer typologies were put such as dropout layers, normalizations and additional activation functions. To have a better understanding of the other model architectures let us see them through a comparison with respect to the initial existing models seen at the beginning of Sections 4.1.

- **v3 - CONV-R** - Convolutional, *ReLU*, Convolutional, ReLU, Linear, Linear;
- **v4 - CONV-D** - Convolutional, Convolutional, ReLU, *Dropout(p=0.2)*, Linear, Linear;
- **v5 - CONV-RDD** - Convolutional, *ReLU*, *Dropout(p=0.1)* Convolutional, ReLU, *Dropout(p=0.1)*, Linear, Linear;
- **v6 - CONV-LN** - Convolutional, Convolutional, ReLU, *Layer normalization*, Linear, Linear;
- **v7 - CONV-GN** - Convolutional, Convolutional, ReLU, *Group normalization*, Linear, Linear;

- **v8 - CONV-RDN** - Convolutional, *ReLU*, *Dropout(p=0.1)*, Convolutional, ReLU, *Dropout(p=0.1)*, *Layer normalization*, Linear, Linear;
- **v9 - LIN-D** - Linear, *Dropout(p=0.2)*, Linear;
- **v10 - LIN-R** - Linear, Linear, *ReLU*;
- **v11 - LIN-C** - *Linear*, *Convolutional*, *ReLU*, Linear, Linear;

Versions from 3 to 8 are inspired to the model version number 1 (CONV), while 9 to 11 are inspired to version 0 (LIN). We can see the variations from the initial version written with *italic* text. New layers were added by focusing on where to insert them and which layer type to put in that point. First of all, for model version CONV a critical point was in between the two convolutional layers. Furthermore, a good point to place the dropout layer, in geneeral, is after the activation function, so it was placed there. Layer typology, instead, was made quite random instead; an approach that is usually taken on model architecture experiments. Training strategy was the same as before, employing 15 and 50 epochs and different learning rates. After the longer training a comparison with each new version loss curves were compared with those of the model from which it came from.

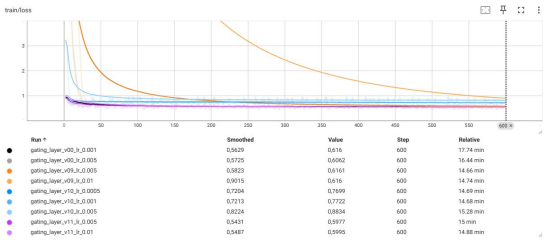

Figure 4.30: LIN derived models training losses, 50 epochs

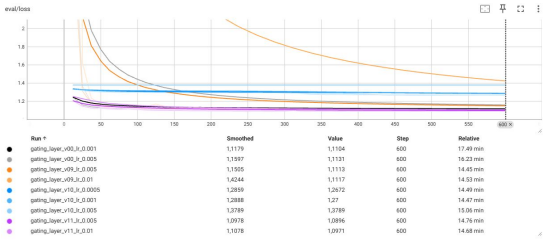

Figure 4.31: LIN derived models validation losses, 50 epochs

Results (Fig. 4.30, 4.31) for LIN-D and LIN-R were a bit higher than LIN, so I expect similar results from them. Despite that LIN-R had almost flat curves in validation part, so I do not expect any good results from that model. LIN-C, instead, had lower losses with respect to LIN for the training and validations parts. These curves are also related to higher learning rates (0.01 and 0.005).

Speaking about CONV related models (Fig. 4.32, 4.33) CONV-R and CONV-RDD had a remarkable aspect that consisted in a gap inside the training and validation losses by passing from learning rate 0.01 to 0.005. CONV-RDN, which has a similar architecture, has a lower gap and this could be due to the normalization layer that there is in this last model. By the way, all these losses end up in the

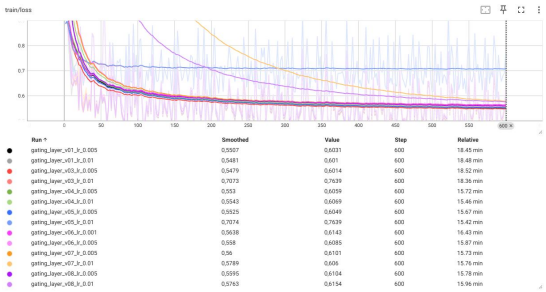

Figure 4.32: CONV derived models training losses, 50 epochs

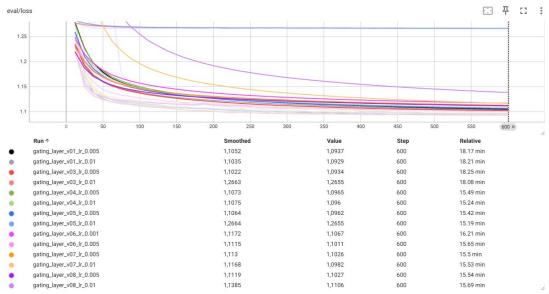

Figure 4.33: CONV derived models validation losses, 50 epochs

region of CONV loss. CONV-D and CONV-GN have similar loss curves to CONV and no particular aspects need to be highlighted. CONV-LN, finally, has losses that are further from any of the previous models, but they still stand in a reasonable region.

After having seen and discussed the model versions trained for 50 epochs let us have a look at if we are getting what we expect by the waveform analysis. In order to give a comparison of the models performances results are related to only one sample of those shown before, post-processed with minmax rescaling. Results related to other samples and different filtering methodologies will be analyzed in Section 4.7. For comparison purposs, in Figure 4.34 the original sample is followed by sonification results provided by the best LIN (2nd and 3rd sample) and CONV (4th and 5th sample) models, so here there ar baseline related results.

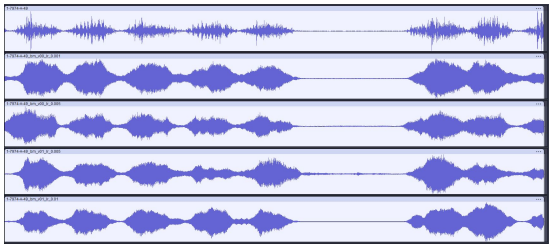

Figure 4.34: LIN and CONV 1-7974-A-49 sonifications

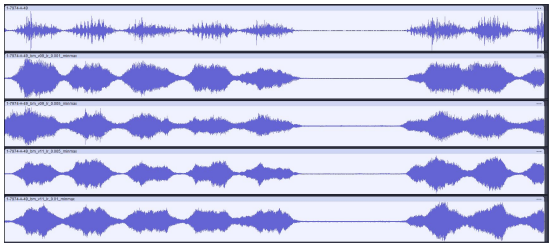

Figure 4.35: LIN-D and LIN-C 1-7974-A-49 sonifications

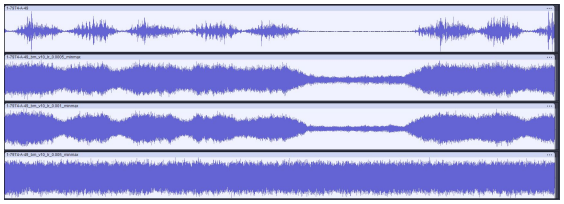

Figure 4.36: LIN-R 1-7974-A-49 sonifications

In Fig. 4.35 and 4.36 we have the sonification results of the versions derived from gating layer LIN. As we could expect, from the loss curves seen before, LIN-D and LIN-C, whose are similar to LIN, approximate well the audio waveform carrying a small amount of noise in between the peaks (silence part). LIN-C seems to better mimic the original sample behavior with more precise heights. This could be due to the fact that LIN-C adds, among the additional layers, also a convolutional one, taking advantage of this. LIN-R, on the other hand, performs poorly as we also expected. This version with learning rate 0.005 has a loss curve that is too much higher, and the waveform is reconstructed with basically only noise. Learning rates 0.001 and 0.0005 are better since the sonification is not like the previous one, but the audios remain very disturbed.

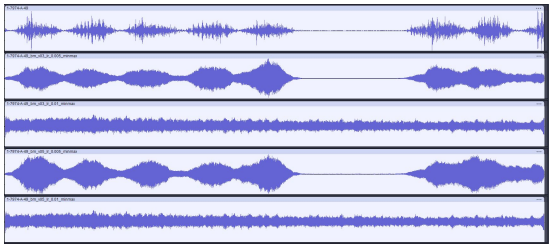

Figure 4.37: CONV-R and CONV-RDD 1-7974-A-49 sonifications

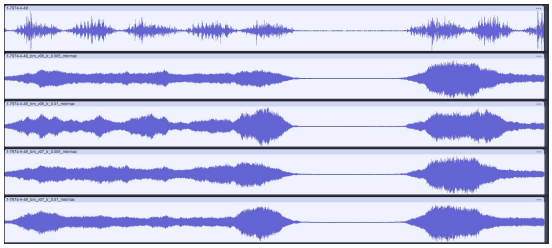

Figure 4.38: CONV-LN and CONV-GN 1-7974-A-49 sonifications

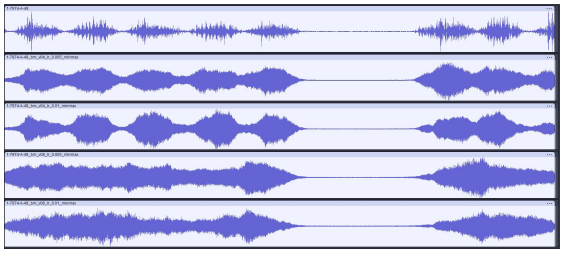

Figure 4.39: CONV-D and CONV-RDN 1-7974-A-49 sonifications

Figure 4.37 are related to CONV-R and CONV-RDD, those models that presented a gap in between the training losses. This can be noted in the results. Both models, in fact, are characterized by a reconstruction that is made with just noise for the learning rate=0.01. It then gives a more understandable one when passing to the next lower learning rate, with a good silence part that is not so disturbed, but, as the loss curve says, CONV version is still better being better in reducing the waveform height in between the peaks.

In Fig. 4.38 we have the results produced by models CONV-LN and CONV-GN, the versions in which I experimented with the type of normalization. We can clearly see that changes due to its type variation are so minimal but these were the expected sonification after looking at the training and validation losses. Despite the losses being similar to the of CONV the sonifications are not so precise bringing two main peaks instead of 7 and a half. By the way, they grant a good behavior on silence part.

Finally, from Figure 4.39 we can see the effect of the dropout layer on CONV-D, that does not change so much the CONV baseline results shown in Figure 4.34. CONV-RDN, finally, shows poor results with two main peaks as CONV-LN and CONV-GN instead of seven and a half crests. This could be due to the normalization layer that could be not so suitable for the task we want to solve. The silence part is good, but this can not overcome the audios obtained from some other versions.

From these examples we can conclude that the model, as it should be, is the principal cause of the audio output, and the post-processing rescaling and filter could be helpful sometimes, especially with those models that are not as precise in terms of loss curve. To the best models LIN and CONV we can add LIN-C, LIN-D and CONV-D that had a good comparison in terms of losses and sonification results.

Unfortunately, this is true for the majority of audio samples. But there are some classes where these versions just can not perform in a good way. This is the case of continuous sounds like washing machines, people clapping, trains or planes that have not a clear division between sound and silence parts. We have an example in Figure 4.40 where are shown the waveform produced by gating layer best versions derived from a washing machine recording.

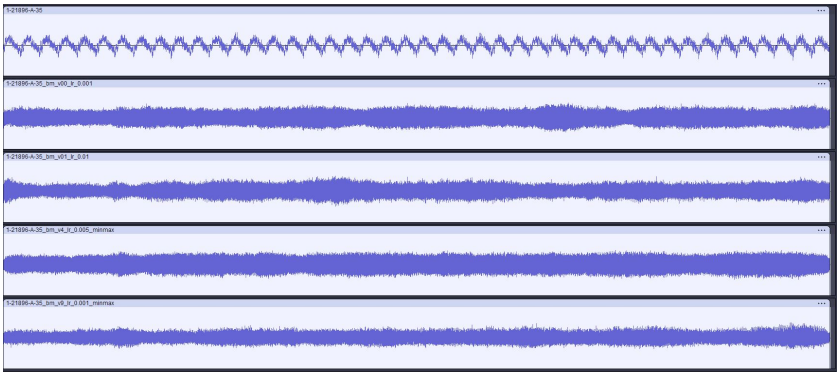

Figure 4.40: LIN, CONV, CONV-D, LIN-D, LIN-C 2-262579-A-45 sonification results after 50 epochs, audio rescaling

4.6 Non-spectrogram based transformer: HuBERT

As last experiment, the entire procedure was made starting from HuBERT embeddings, that does not process the audio spectrogram during its decision process (see Section 2.5.2 for more details about that). In this way a more general analysis on this novel mapping approach would have been set.

First of all, as done for AST, HuBERT finetuning was needed, since the baseline which I used was trained on 960 hours of librispeech, and so speech audios. Some already finetuned versions of HuBERT on ESC50 were found, but I decided to fine-tune it by myself using the same parameters, except for the number of epochs and, after 50 epochs, I obtained a validation accuracy of 83.5% at epoch 50.

After that, th finetuned model was imported and used to compute its embeddings. This procedure was a bit different with respect to the AST model since HuBERT does not have a patch embedder, but just a few lines of code before the model encoder in the class named *HubertModel*. Despite that, a custom method was built that returned the embedding representation of the first hidden state of the model.

At this point, gating layer code was modified accordingly to take as input the HuBERT embeddings instead of the AST ones. Consequently, all the versions were trained for the usual 15 and 50 epochs. Training results of all versions are shown in Figures from 4.41 to 4.44.

From graphs, we can clearly see a loss gap in between the learning rate of model versions CONV-R, CONV-RDD, and CONV-RDN. We are not new to these gaps that were already present in the loss given after the training with AST embeddings. This gap is also present in validation losses. We can see that the majority of the model have small value losses overall. Among those with the lower values we can

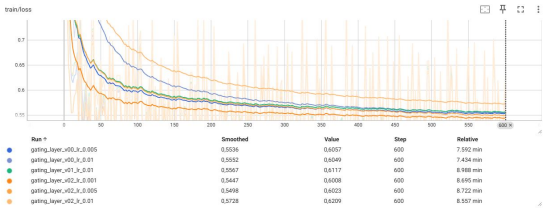

Figure 4.41: LIN, CONV and ATTN best training losses, 50 epochs

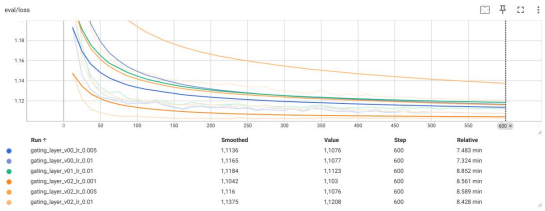

Figure 4.42: LIN, CONV and ATTN best validation losses, 50 epochs

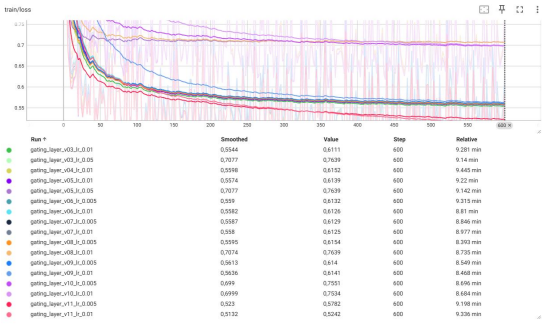

Figure 4.43: CONV-R to LIN-C best training losses, 50 epochs

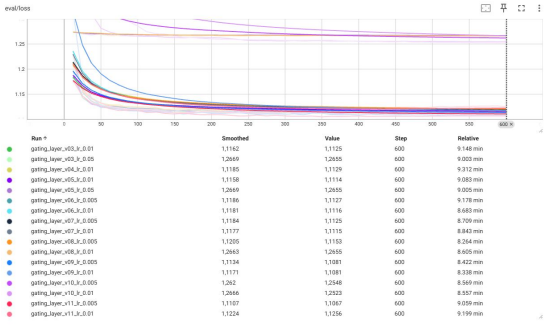

Figure 4.44: CONV-R to LIN-C best validation losses, 50 epochs

find LIN, ATTN, LIN-C, CONV and CONV-R with the smallest learning rates while the worst is still version LIN-R. We can further note that some of the best models were also the best model versions after the training with AST embedding too. In this list we can find LIN, LIN-C and CONV. An important enhancement was made by gating layer ATTN that reached the second lowest loss value, so we should expect great improvements in its sonifications from those obtained with AST embeddings. Other models should perform in line with the previously analyzed sonifications.

In order to address this aspect, we are going to analyze the results produced by these gating layer versions sonifying HuBERT embeddings coming from the three audios already analyzed in Section 4.4.

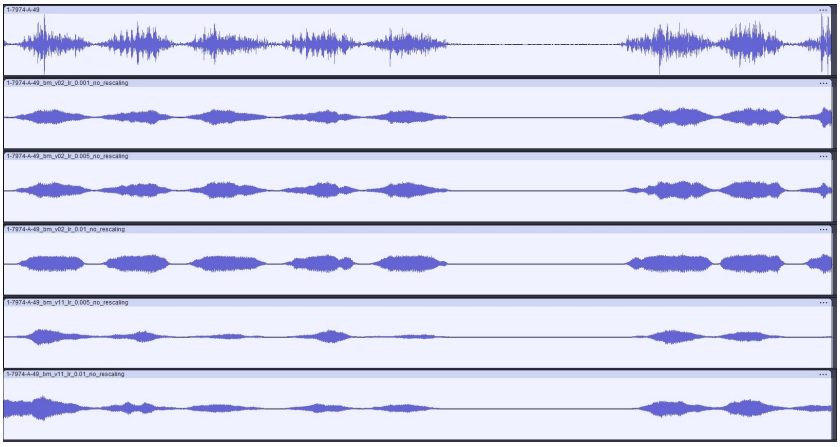

Figure 4.45: HuBERT sonifications from best models versions (ATTN and LIN-C), no rescaling

Image 4.45 shows best models sonification results. We can see how a post sonification rescaling is be useful also with this model embeddings. So, let us discuss the rescaled results associate with the all the models versions related to the best learning rates w.r.t. their loss value.

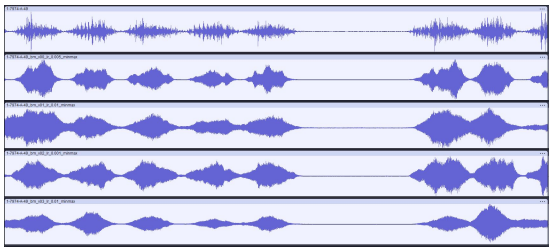

Figure 4.46: HuBERT sonifications from gating layer LIN, CONV, ATTN, CONV-R best learning rates, minmax rescaling

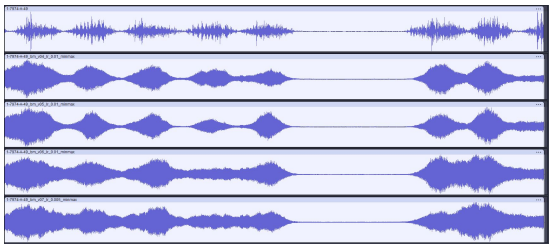

Figure 4.47: HuBERT sonifications from gating layer CONV-D, CONV-RDD, CONV-LN, CONV-GN best learning rates, minmax rescaling

From figures 4.46, 4.47 and 4.48 we can see how the reconstructions are generally better than those obtained from AST embeddings sonifications with all the gating layer versions. As we expected LIN-C and ATTN performed in a good way providing an output with well distinct peaks and a silence without background noise. LIN outputs a waveform with high definition crests that approximates the original sample in a remarkable way. LIN-R, on the other hand, has background noise before and after each crests and also on the silence part. CONV-RDD an CONV-RDN gives results similar to CONV-R due to the similar gating layer architecture,

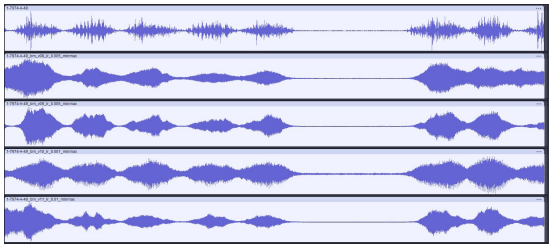

Figure 4.48: HuBERT sonifications from gating layer CONV-RDN, LIN-D, LIN-R, LIN-C best learning rates, min-max rescaling

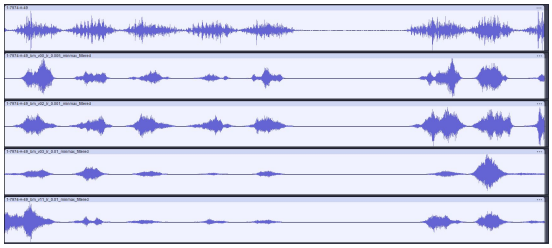

Figure 4.49: HuBERT sonifications from best gating layer LIN, ATTN, CONV-R, LIN-C minmax rescaling and filtering

while CONV-LN and CONV-GN output a discrete result.

In general these are good results by their own and they do not need any filtering, but, to have a comparison with the previous section in Figure 4.49 there are the square filtered sonifications deriving from the best gating layer versions.

We can note that the filtering approach on these sonifications is anything but good. The crests are radically reduced, transforming the good initial audio quality. The filtering technique ruins the output provided by the stand alone models, so should be avoided on sonifications obtained from HuBERT embeddings.

Finally, let us analyze those other particular samples seen in previous sections. From Figures 4.50 and 4.51 we can note how the models can better reconstruct audios with lower volumes without affecting the output quality. CONV-R has some troubles in the 2-120587-A- hen reconstruction leading to a sonification that is not particularly good.

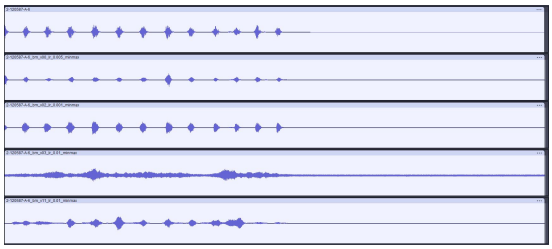

Figure 4.50: HuBERT sonifications of sample 2-120587-A-6 from gating layer LIN, ATTN, CONV-R, LIN-C minmax rescaling

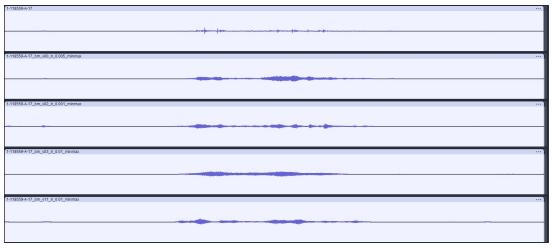

Figure 4.51: HuBERT sonifications of sample 1-118559-A-17 from gating layer LIN, ATTN, CONV-R, LIN-C minmax rescaling

Figures 4.50, 4.51 and 4.52 show how remarkably gating layers trained on

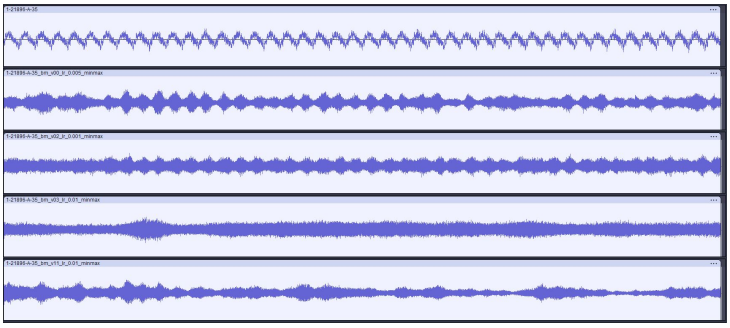

Figure 4.52: HuBERT sonifications of sample 1-21896-A-35 from gating layer LIN, ATTN, CONV-R, LIN-C minmax rescaling

HuBERT embeddings reconstruct better also those continuous sounds; this model type versions provide an output with continuous volume changes that mimic the original sample in a good way, very different to the sonification in Figure 4.40. This aspect could be due to the fact that while AST uses two tokens for the final predictions, while HuBERT exploits a mean of the hidden state values. Since AST embeddings were used without those tokens, as they could have biased the gating layer mapping and, so, the sonification process, these tensors had lower information with respect to the ones belonging to HuBERT. So, the logic behind the model we want to explain could impact on the quantity of information carried by its internal representations.

4.7 Qualitative and quantitative analysis

In order to attribute an objective evaluation for the sonification process, a final quantitative analysis was performed. By using KAD and FAD metrics the sonifications obtained from AST and HuBERT model embeddings were evaluated; the sonifications were post processed with the two main methods: minmax rescaling and the rescaling combined with the squared filtering. KAD toolkit described in Section 3.0.4 was used, and that allowed to compute the two metrics by simply running a command. Then, a qualitative analysis was performed on the same samples; finally the two evaluation methods were compared in order to say if they were providing trustworthy results.

Objective metrics are shown in Figures 4.53, 4.54, 4.55, 4.56. For each model version, we have a point mapped to the value of that KAD metric computed on its output samples. A dotted horizontal line meets the lower KAD point that is related to the best gating layer version. Vertical lines are there just to better connect points with related model versions. KAD metric was computed by comparing one sample at a time with the original sound from ESC50 dataset. As we can see in

the original column, the lower the score, the better the model. This column, in fact, is not related to any obtained sonification, but is just the KAD computed by comparing the original audio with itself. While images provide an overall idea that allows a fast comparison between the model versions, KAD scores are not so clear due to the large scale. To address that issue four tables with the top 5 best gating layer versions for each model embedding type+post_processing method are provided (see Tables from 4.1 to 4.4). Tables provide also a view on FAD metric and, as we could expect from its set dimension issues, it gives useless values. For this reason, I relied on KAD score for the sonification objective evaluation.

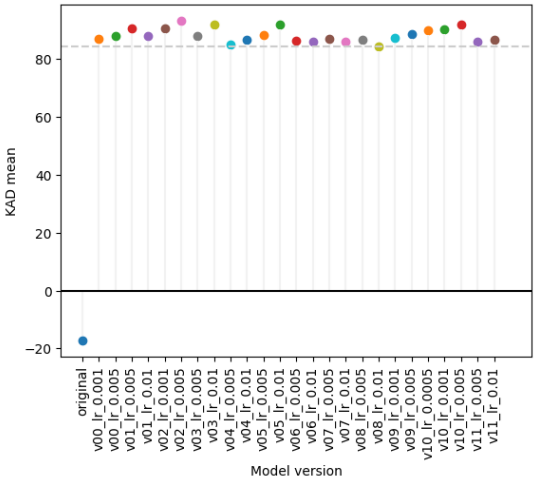

Figure 4.53: KAD metric computed on AST sonifications with post processing rescaling

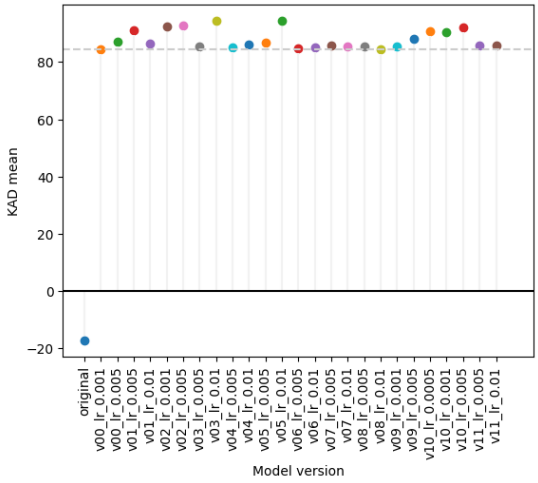

Figure 4.54: KAD metric computed on AST sonifications with post processing rescaling and filtering

Model version	KAD score	FAD score
CONV-RDN lr=0.01	84.546934	NaN
CONV-D lr=0.005	85.093417	NaN
CONV-LN lr=0.01	86.005913	-inf
CONV-GN lr=0.01	86.120367	-1.815338e+30
LIN-C lr=0.005	86.124735	3.935136e+30

Table 4.1: Top 5 Gating Layer versions trained on AST embeddings with audio rescaling as post processing

From the first two figures, we can see that model version ATTN has the worst score overall and, also the other one is not among the best values. This also happens

Model version	KAD score	FAD score
CONV-RDN lr=0.01	84.474238	2.191557e+31
LIN lr=0.001	84.503346	6.116645e+30
CONV-LN lr=0.005	84.741687	-3.167567e+29
CONV-LN lr=0.01	85.066978	1.710227e+30
CONV-D lr=0.005	85.077784	-2.925982e+30

Table 4.2: Top 5 Gating Layer versions trained on AST embeddings with audio rescaling and filtering as post processing

for CONV-R and CONV-RDD lr=0.01 and LIN-R, which were the models that also provided bad results for the initial qualitative tests. From the tables, instead, we can find that what were the best versions at first sight as CONV-R, LIN-D and LIN-C, are not numerically the bests that are surpassed from CONV-RDN, CONV-LN and CONV-GN that were not so particular during previous tests. This could be due to the fact that this analysis were computed on 77 samples, with at least one sample for each class. So maybe I took a "lucky" sample that the model could reconstruct in a good way, or simply I made some mistake in the evaluation since I am not an expert of audio similarity evaluation. Furthermore, we can say that the theory on audio filtering was correct: it could enhance, in general, models outputs. We can see that by the lower values in Table 4.53 w.r.t. those in Table 4.54. Furthermore, we can observe that CONV-RDN with learning rate=0.01 remains the best model with both rescaling and rescaling+filtering post processing methods. From this, we can conclude its ability into audio reconstruction.

On the other hand, if we look at the other figures and tables we can see how the gating layer architectures performs in mapping HuBERT embeddings. From the models dots we can see that most of the model performs better with this set of embeddings, giving proof that the HuBERT hidden state carries more information than the AST internal representation.

Model version	KAD score	FAD score
ATTN lr=0.005	83.136840	-3.083930e+30
LIN lr=0.01	83.801301	inf
LIN lr=0.005	83.811109	-6.868389e+30
ATTN lr=0.001	83.829551	-inf
LIN-D lr=0.01	84.218821	-7.106036e+30

Table 4.3: Top 5 Gating Layer versions trained on HuBERT embeddings with audio rescaling as post processing

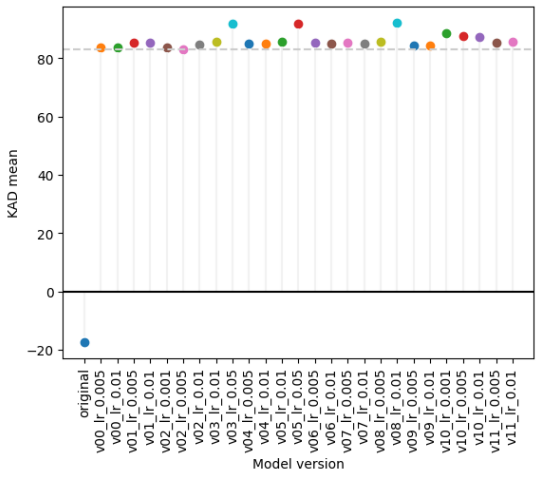

Figure 4.55: KAD metric computed on HuBERT sonifications with post processing rescaling

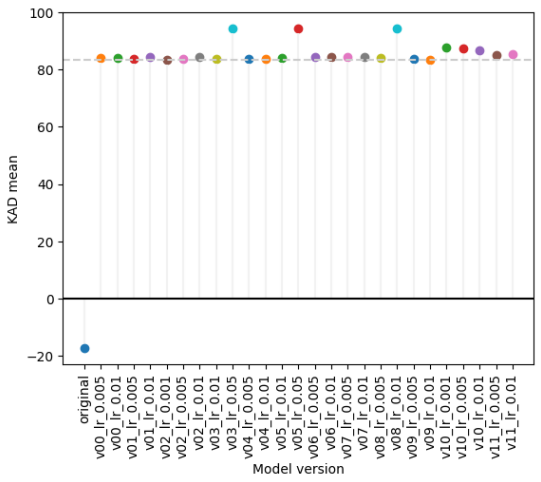

Figure 4.56: KAD metric computed on HuBERT sonifications with post processing rescaling and filtering

Model version	KAD score	FAD score
LIN-D lr=0.01	83.508207	1.205736e+31
ATTN lr=0.001	83.511286	2.044940e+29
ATTN lr=0.005	83.689330	inf
CONV-D lr=0.01	83.707093	9.274549e+3
CONV lr=0.005	83.797345	inf

Table 4.4: Top 5 Gating Layer versions trained on HuBERT embeddings with audio rescaling and filtering as post processing

We can observe how ATTN passes from the worst to the best gating layer being two times into the top 5 with learning rates 0.005 and 0.001. LIN also has this characteristic followed by LIN-D. This demonstrates what was said before, where really simple models (fill linear or similar) or very complex ones (attention based) are the best to compute this task. Furthermore, also the other architectures work well by being less far from the horizontal best KAD line than those points regarding AST gating layer versions. This time, filtering has a destructive tendency, since the associated KAD scores are higher than those post-processed by simply rescaling method. Moreover, also KAD scores for models that are enhanced with this procedure are not so far from the values without this post-processing, so it should be avoided the usage of this technique within these specific model versions.

Finally, these results made possible to choose the best performing models and make a subjective qualitative analysis on them. As already said, I assigned a mark on each of the 77 sonifications provided by the four configurations in which the gating layer model was used. These configurations were:

1. sonify AST embeddings and then minmax rescaling;
2. sonify AST embeddings and then minmax rescaling+filtering;
3. sonify HuBERT embeddings and then minmax rescaling;
4. sonify HuBERT embeddings and then minmax rescaling+filtering.

Results were annotated on *.csv* files that are available in the Google Drive space. Here, I will report marks means for each model version. Results are in Tables from 4.5 to 4.8.

AST embeddings, sonification rescaling		
Model version	Visual	Audio
LIN LR=0.001 - 11th	3,883117	**3,792208**
CONV LR=0.01 - 13th	**3,948052**	3,727273
ATTN LR=0.001 - 23th	1,532468	1,246753
CONV-D LR=0.005 - 2nd	3,844156	3,805195
CONV-RDN LR=0.01 - 1st	2,740260	2,285714

Table 4.5: AST embeddings sonification post-processed with minmax rescaling

AST embeddings, sonification rescaling AND filtering		
Model version	Visual	Audio
LIN LR=0.001 - 2nd	**3,961039**	**3,909091**
CONV LR=0.01 - 14th	3,623377	3,350649
ATTN LR=0.001 - 22th	1,610390	1,220779
CONV-LN LR=0.005 - 3rd	3,051948	2,649351
CONV-RDN LR=0.01 - 1st	2,545455	2,142857

Table 4.6: AST embeddings sonification post-processed with minmax rescaling and squared filtering

HuBERT embeddings, sonification rescaling		
Model version	Visual	Audio
LIN LR=0.01 - 2nd	4,649351	**4,584416**
CONV LR=0.005 - 13th	3,441558	3,116883
ATTN LR=0.005 - 1st	**4,662338**	4,545455
CONV-GN LR=0.01 - 5th	3,415584	3,194805
LIN-D LR=0.01 - 8th	4,376623	4,259740

Table 4.7: HuBERT embeddings sonification post-processed with minmax rescaling

HuBERT embeddings, sonification rescaling AND filtering		
Model version	Visual	Audio
LIN LR=0.01 - 9th	3,649351	3,714286
CONV LR=0.005 - 5th	2,896104	2,610390
ATTN LR=0.001 - 2nd	**3,935065**	**3,870130**
CONV-D LR=0.01 - 4th	2,948052	2,792208
LIN-D LR=0.01 - 1st	3,454545	3,363636

Table 4.8: HuBERT embeddings sonification post-processed with minmax rescaling and squared filtering

Models are ordered w.r.t. their version number. The number at their right is the position among their category classification. We can observe that the supposition on the quantity of information carried from the HuBERT model embeddings is shown also in this type of analysis. Tables 4.7 and 4.8 are characterized from higher mean values if compared to the other two. Furthermore, the filtering approach enhances gating layer versions that were not performing so well after a simple sonification rescale. On the other hand, this technique make perform worse models that were actually good in the configuration without filtering. E.g. in the sonification of AST embeddings LIN with learning rate=0.001 passes from 11th position to the 1st or, with HuBERT embeddings, LIN-D becomes the first model when its results are filtered surpassing ATTN that were the most promising one. Moreover, ATTN performances are lower, but they are still the best. Objective analysis is relatable with the qualitative one only sometimes. We can do this comparison if we compare the classification grades in each tables with the related Visual/Audio score. This is the case of ATTN in configuration 1 or ATTN and LIN in configuration 3. Despite that, especially for gating layer CONV-RDN mapping AST embeddings, the model, to my advice, produce sonifications that are not so good. This could be due to the way the KAD score is performed, what are the audio features taken into consideration, or just simply from the fact that I am not an expert and someone

else could give totally different grades to the sonifications. Finally, we can observe some discrepancies between the assigned visual and the audio mark. This is due to audios that are easier to classify in a visual way, while others are better classifiable by listening to them, capturing different features accordingly to the evaluation method. The most common cases were loud and almost silent audios.

Chapter 5

Conclusions

5.1 Conclusions

This thesis works focused on mapping hidden state representation of audio transformer models to the latent space of a generative audio model in order to sonify them. While the gating layer model already existed, the fact of using it to retrieve the embedding sonification is a novel idea, that could open the path to new research techniques focused on the embedding interpretation, a representation that was never explored in this way before.

Due to its limited dimensions, the gating layer can be easily trained on an HPC system by taking a reasonable amount of time to solve this task. With these gating layer versions significant results were obtained, where the reconstructed audio followed the majority of input waveforms in a good way. This method was tested on audio transformers that employed spectrograms or simply used the raw waveform to compute their embeddings, providing a general method for both architectures. The output of the sonified gating layer translation were post processed with different techniques, that were not always worthy, but that sometimes could leverage sonification outputs. Finally, these results were given subjectively and objectively leading to an a personal and on an objective evaluation.

5.2 Future works

Despite the remarkable results reached, a lot of work has still to be done in this particular data domain, since no one investigated on it before. First of all, new version of the model could be tried with new hyperparameters, loss types and layers. By what we have seen the simpler the model, the better the results, but this depends from which model we want to sonify and I did not tried any larger version, so i can not confirm that! So, new models could be used as starting point to get the

embeddings and different could be used in order to obtain the sonification results. Moreover, other filtering methods and other metrics could be used, considering also to organize a user study to enhance results of qualitative analysis.

Furthermore, this methodology can be used in addition to existing approaches such as those based on gradient flow to give a more precise attribution of which part of the input is used in the decision process of the model.

In short, a lot of work has still to be done, but we hope to have given a new view, a new starting point on a new possible explainability method that could inspire other researchers in the future.

Appendix A

Appendix A

Encoder blocks:

1. 3x residual units: refer to the next list to see this unit expanded.
2. Snake1d activation: since the activation functions maintain inputs values in a certain range this is similar to a value clipping, so no changes have been made on the axes.
3. Weighted norm 1D-Convolution: no changes on the temporal dimension.

Residual unit:

1. Snake1d activation: as we said before, this does not change the temporal orientation.
2. Weighted norm 1D-Convolution: no changes on the temporal dimension.
3. Snake1d activation: as we said before, this does not change the temporal orientation.
4. Weighted norm 1D-Convolution: no changes on the temporal dimension.

Bibliography

[1] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL]. URL: https://arxiv.org/abs/2302.13971 (cit. on pp. 1, 5).

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805 (cit. on pp. 1, 5, 11).

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. «Attention Is All You Need». In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (cit. on p. 2).

[4] Yuan Gong, Yu-An Chung, and James Glass. *AST: Audio Spectrogram Transformer*. 2021. arXiv: 2104.01778 [cs.SD]. URL: https://arxiv.org/abs/2104.01778 (cit. on pp. 5, 11, 14).

[5] Alexey Dosovitskiy et al. «An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale». In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929 (cit. on p. 5).

[6] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. «BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension». In: *CoRR* abs/1910.13461 (2019). arXiv: 1910.13461. URL: http://arxiv.org/abs/1910.13461 (cit. on p. 5).

[7] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*. 2020. arXiv: 2006.11477 [cs.CL]. URL: https://arxiv.org/abs/2006.11477 (cit. on pp. 11, 15).

[8] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. *HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units*. 2021. arXiv: 2106.07447 [cs.CL]. URL: https://arxiv.org/abs/2106.07447 (cit. on pp. 11, 15).

[9] Sanyuan Chen et al. *WavLM: Large-Scale Self-Supervised Pre-Training for Full Stack Speech Processing*. Oct. 2022. DOI: 10.1109/jstsp.2022.3188113. URL: http://dx.doi.org/10.1109/JSTSP.2022.3188113 (cit. on p. 11).

[10] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. *Robust Speech Recognition via Large-Scale Weak Supervision*. 2022. arXiv: 2212.04356 [eess.AS]. URL: https://arxiv.org/abs/2212.04356 (cit. on p. 11).

[11] Sara Atito Ali Ahmed, Muhammad Awais, Wenwu Wang, Mark D. Plumbley, and Josef Kittler. «ASiT: Local-Global Audio Spectrogram Vision Transformer for Event Classification». In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 32 (2024), pp. 3684–3693. ISSN: 2329-9304. DOI: 10.1109/taslp.2024.3428908. URL: http://dx.doi.org/10.1109/TASLP.2024.3428908 (cit. on p. 11).

[12] Yuan Gong, Cheng-I Lai, Yu-An Chung, and James R. Glass. «SSAST: Self-Supervised Audio Spectrogram Transformer». In: *ArXiv* abs/2110.09784 (2021). DOI: 10.1609/aaai.v36i10.21315 (cit. on p. 11).

[13] Xinhao Mei, Xubo Liu, Qiushi Huang, Mark D. Plumbley, and Wenwu Wang. *Audio Captioning Transformer*. 2021. arXiv: 2107.09817 [eess.AS]. URL: https://arxiv.org/abs/2107.09817 (cit. on p. 11).

[14] Heinrich Dinkel, Zhiyong Yan, Yongqing Wang, Junbo Zhang, Yujun Wang, and Bin Wang. *Streaming Audio Transformers for Online Audio Tagging*. 2024. arXiv: 2305.17834 [cs.SD]. URL: https://arxiv.org/abs/2305.17834 (cit. on p. 11).

[15] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML]. URL: https://arxiv.org/abs/1312.6114 (cit. on p. 12).

[16] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML]. URL: https://arxiv.org/abs/1406.2661 (cit. on p. 12).

[17] Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. *HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis*. 2020. arXiv: 2010.05646 [cs.SD]. URL: https://arxiv.org/abs/2010.05646 (cit. on p. 12).

[18] Rithesh Kumar, Prem Seetharaman, Alejandro Luebs, Ishaan Kumar, and Kundan Kumar. *High-Fidelity Audio Compression with Improved RVQGAN*. 2023. arXiv: 2306.06546 [cs.SD]. URL: https://arxiv.org/abs/2306.06546 (cit. on pp. 12, 15).

[19] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015. arXiv: 1503.03585 [cs.LG]. URL: https://arxiv.org/abs/1503.03585 (cit. on p. 13).

[20] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. *Diffusion Models: A Comprehensive Survey of Methods and Applications*. 2024. arXiv: 2209.00796 [cs.LG]. URL: https://arxiv.org/abs/2209.00796 (cit. on p. 13).

[21] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. «Layer-wise relevance propagation: an overview». In: *Explainable AI: interpreting, explaining and visualizing deep learning* (2019), pp. 193–209 (cit. on p. 13).

[22] Hila Chefer, Shir Gur, and Lior Wolf. *Transformer Interpretability Beyond Attention Visualization*. 2021. arXiv: 2012.09838 [cs.CV]. URL: https://arxiv.org/abs/2012.09838 (cit. on p. 13).

[23] Isidoros Perikos and Athanasios Diamantopoulos. «Explainable Aspect-Based Sentiment Analysis Using Transformer Models». In: *Big Data and Cognitive Computing* 8.11 (2024). ISSN: 2504-2289. DOI: 10.3390/bdcc8110141. URL: https://www.mdpi.com/2504-2289/8/11/141 (cit. on p. 13).

[24] Catherine Yeh, Yida Chen, Aoyu Wu, Cynthia Chen, Fernanda Viégas, and Martin Wattenberg. *AttentionViz: A Global View of Transformer Attention*. 2023. arXiv: 2305.03210 [cs.HC]. URL: https://arxiv.org/abs/2305.03210 (cit. on p. 13).

[25] Alican Akman and Björn W. Schuller. «AttHear: Explaining Audio Transformers Using Attention-Aware NMF». In: *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2024), pp. 7015–7019. DOI: 10.1109/ICASSP48485.2024.10447390 (cit. on p. 14).

[26] Paolo Fantozzi and Maurizio Naldi. «The Explainability of Transformers: Current Status and Directions». In: *Comput.* 13 (2024), p. 92. DOI: 10.3390/computers13040092 (cit. on p. 14).

[27] innnky. *Descript Audio VAE*. https://github.com/innnky?tab=repositories [Last commit: (April 02, 2024)]. 2023 (cit. on p. 15).

[28] Karol J. Piczak. «ESC: Dataset for Environmental Sound Classification». In: *Proceedings of the 23rd Annual ACM Conference on Multimedia*. Brisbane, Australia: ACM Press, Oct. 13, 2015, pp. 1015–1018. ISBN: 978-1-4503-3459-4. DOI: 10.1145/2733373.2806390. URL: http://dl.acm.org/citation.cfm?doid=2733373.2806390 (cit. on p. 18).

[29] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on p. 23).

[30] Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi. *Fréchet Audio Distance: A Metric for Evaluating Music Enhancement Algorithms*. 2019. arXiv: 1812.08466 [eess.AS]. URL: https://arxiv.org/abs/1812.08466 (cit. on p. 24).

[31] Yoonjin Chung, Pilsun Eu, Junwon Lee, Keunwoo Choi, Juhan Nam, and Ben Sangbae Chon. *KAD: No More FAD! An Effective and Efficient Evaluation Metric for Audio Generation*. 2025. arXiv: 2502.15602 [cs.SD]. URL: https://arxiv.org/abs/2502.15602 (cit. on pp. 24, 25).