Interpretable Machine Learning for Machine Listening

Saumitra Mishra

Submitted in partial fulfilment of the requirements
of the Degree of Doctor of Philosophy

School of Electronic Engineering and Computer Science
Queen Mary University of London
United Kingdom

January 2020

Statement of Originality

I, Saumitra Mishra, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the university has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

Signature: Saumitra Mishra
Date: 08/01/2020

Details of collaboration and publications: see Section 1.5

Abstract

Recent years have witnessed a significant interest in interpretable machine learning (IML) research that develops techniques to analyse machine learning (ML) models. Understanding ML models is essential to gain trust in their predictions and to improve datasets, model architectures and training techniques. The majority of effort in IML research has been in analysing models that classify images or structured data and comparatively less work exists that analyses models for other domains. This research focuses on developing novel IML methods and on extending existing methods to understand machine listening models that analyse audio. In particular, this thesis reports the results of three studies that apply three different IML methods to analyse five singing voice detection (SVD) models that predict singing voice activity in musical audio excerpts.

The first study introduces SoundLIME (SLIME), a method to generate temporal, spectral or time-frequency explanations for predictions of any machine listening model. The study involves applying SLIME to analyse the trustworthiness of three SVD models for some carefully selected instances. Results indicate that SLIME effectively identifies that the binary decision tree model is untrustworthy and may not generalise. Moreover, the study analyses the behaviour of SLIME for two input parameters and the results suggest that the choice of suitable values for those parameters is essential to generate reliable explanations from SLIME.

The second study introduces a novel method to perform activation maximisation (AM), a technique that synthesises examples that maximally activate the components (neurons, layers) of a deep neural network (DNN). The method uses a generative adversarial network as a prior in the AM pipeline. The study involves applying the method to synthesise examples for understanding two DNN-based SVD models. Examples that the method synthesises for the output layer neurons in both the models exhibit the presence of vocal and non-vocal characteristics for their respective inputs suggesting that those neurons have learnt to detect high-level class concepts. The study also introduces and demonstrates a method for quantitatively selecting suitable values for AM hyper-parameters. The observation about the presence of class characteristics in the synthesised examples is further supported by the results of an online perceptual study involving 23 participants.

The third study demonstrates that feature inversion, a method to invert features (handcrafted or learned) back to the input space, is an effective method

for explaining DNN predictions. The study also involves applying feature inversion to understand features that each layer of the DNN-based SVD model preserves. The qualitative analysis of inverted representations corresponding to the deepest hidden layer suggests that the representations corresponding to the vocal and non-vocal excerpts contain energy mostly in the higher and lower frequency regions, respectively.

In conclusion, this thesis contributes to IML research by developing novel post-hoc analysis methods and to machine listening research by providing effective tools for investigating and understanding machine listening models. Hopefully, insights from model analysis will assist in developing trustworthy ML models with better generalisation capabilities.

Acknowledgements

This research would not have been possible without the help and support of many people. First and foremost, I would like to thank my supervisor **Simon Dixon** for giving me the opportunity to work with him. Over the period of four years, Simon has been a great mentor, and his helpful suggestions have assisted me in understanding the different aspects of PhD research. I am very grateful to him for giving me the freedom to choose my research topic and for proofreading the manuscripts.

I also want to thank my supervisor **Bob L. Sturm** for introducing me to the field of interpretable machine learning, for patiently answering my queries, and helping me in improving my technical writing skills. His paper on "Horses" in machine listening has been a great source of inspiration to me.

I am very grateful to my supervisor **Emmanouil Benetos** for agreeing to supervise this research during the last years of the PhD. His excellent guidance, continuous support, and approachability for technical and other discussions have helped me to accomplish the research goals.

Many thanks to everyone in **C4DM** for creating a great collaborative research environment that helped me in clarifying my doubts and in coming up with new research ideas.

I would also like to thank **Peter Flach** and **Sebastian Stober** for examining this thesis and for providing very helpful and insightful comments for the final version of this thesis.

I am grateful to my father **Pramod Kumar Mishra** and mother **Kavita Mishra** for being my role models, and for motivating me when things went through a difficult phase. I also want to thank my wife **Poornima** for supporting the decision to leave my job for pursuing my dream and for taking care of things when I was busy with the research.

Last, but certainly not the least, I thank my son **Anant** whose jovial nature has kept me relaxed and happy in the times of stress.

Contents

List of Figures

List of Tables

List of Abbreviations

AM	Activation Maximisation
BDT	Binary Decision Tree
CNN	Convolutional Neural Network
Conv	Convolutional
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DNN	Deep Neural Network
ELU	Exponential Linear Unit
FC	Fully-Connected
FFT	Fast Fourier Transform
FN	False Negative
FP	False Positive
GAN	Generative Adversarial Network
ICs	Interpretable Components
IDFT	Inverse Discrete Fourier Transform
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
IML	Interpretable Machine Learning
LIME	Local Interpretable Model-Agnostic Explanations
LSTM	Long Short-Term Memory
MFCC	Mel-Frequency Cepstral Coefficient
MIR	Music Information Retrieval
ML	Machine Learning
MP	Max-Pooling
NRE	Normalised Reconstruction Error
RF	Random Forest
RNN	Recurrent Neural Network
RWC-MDB	Real World Computing Music Database
SA	Sensitivity Analysis
SLIME	Sound LIME
STFT	Short-Time Fourier Transform
SVD	Singing Voice Detection
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
UConv	Up-Convolutional

Chapter 1

Introduction

1.1 Motivation

Machine learning (ML) involves developing computational models using learning algorithms that automatically extract meaningful information from data [Bishop, 2007]. These models have recently demonstrated remarkable success in several complex real-world applications (e.g., image classification, speech recognition) [LeCun et al., 2015, Goodfellow et al., 2016], encouraging their adoption in safety-critical applications (e.g., finance, healthcare). The recent success of ML models is due to several reasons. Some prominent ones include the use of highly expressive models with strong generalisation capability, improved training algorithms, and the availability of large datasets, specialised computing resources (graphics processing units) and open-sourced ML libraries (e.g., Tensorflow) for model training. However, despite all their successes, the majority of ML models are "black-boxes", as theoretically and empirically we know very little about the process by which they form their predictions.

Understanding the behaviour of ML models, referred to as *model interpretability*, is important due to several reasons, one of which is the problem of *incomplete model specifications*. ML models often need to satisfy some auxiliary criteria (e.g., trust, robustness), in addition to achieving the desired prediction performance [Doshi-Velez and Kim, 2017]. However, as quantifying these criteria is often ambiguous, model interpretability provides an alternate way to validate whether ML models satisfy the desired auxiliary criteria. For example, ML models may exploit confounders in a dataset and behave correctly for the wrong reasons [Rodríguez-Algarra et al., 2016]. Such behaviour limits their performance in the real world where such confounders are absent. We can use model interpretability techniques to determine whether confounders influence the predictions of a model and hence, verify its trustworthiness [Guidotti

et al., 2019]. Similarly, researchers have demonstrated that attacking deep neural networks (DNNs) with carefully generated inputs, called "adversarial examples", changes their predictions from correct to incorrect [Szegedy et al., 2014, Goodfellow et al., 2015, Kereliuk et al., 2015]. Such behaviour may be dangerous to a system (e.g., autonomous vehicle) if its decision making depends on DNN predictions [Papernot et al., 2017]. Model interpretability may provide an understanding about the behaviour of DNNs for adversarial examples that in-turn may suggest ways to make DNNs robust against adversarial perturbations. Additionally, model interpretability will help in debugging ML models, fixing dataset artefacts [Ribeiro et al., 2016b], and in refining model architectures [Zeiler and Fergus, 2014]. Thus, there is an urgent need for interpretability in ML models.

Interpretable machine learning (IML) is the field of machine learning that designs methods for model interpretability. There exist two main ways to empirically analyse the behaviour of ML models: by designing inherently interpretable models or by analysing pre-trained models. The first model analysis approach is a promising research direction, and in some applications (e.g., safety-critical applications), it is essential to use interpretable models. However, the interpretable models are often less accurate than the black-box models as generally the training of ML models involves a trade-off between model accuracy and interpretability [Ribeiro et al., 2016a].

The second model analysis approach involves the post-hoc analysis of pre-trained models [Lipton, 2016] and includes methods from two categories [Montavon et al., 2018]. The first category of methods focuses on analysing the global behaviour of ML models, and the insights from the global analysis generalise across input instances. For example, irrespective of the class label associated with an input image, shallow layers of image classification DNNs show sensitivity to low-level structures (e.g., edges) [Zeiler and Fergus, 2014]. There exist several methods for global analysis. For example, activation maximisation synthesises examples in the input space (e.g., images) to maximally activate DNN components (e.g., neurons, layers) [Olah et al., 2017, Erhan et al., 2009].

The second category of post-hoc methods limits the analysis to individual examples (local analysis) and focuses on identifying input features that highly influence model predictions. The vast majority of such methods use variants of sensitivity analysis (SA) to capture the effect of modifying a feature or a group of features on the final prediction. For example, the methods using gradient-based sensitivity analysis for DNNs produce saliency maps that highlight the influence of each input feature towards a prediction [Simonyan et al., 2014]. Similarly, another method generates local explanations by using SA to approximate the non-linear decision boundary in the neighbourhood of an instance by

an interpretable model [Ribeiro et al., 2016b].

Thus, the post-hoc analysis methods assist in gaining insight into the behaviour of ML models without imposing any restrictions on their predictive performance. This model analysis category is the focus of this thesis. Post-hoc interpretability also involves several challenges. For example, some local explanation methods may generate unreliable, inconsistent, and uninterpretable (noisy) explanations [Adebayo et al., 2018, Kindermans et al., 2017, Smilkov et al., 2017]. Moreover, there is a lack of metrics to quantitatively evaluate novel methods and to compare different methods for benchmarking [Doshi-Velez and Kim, 2017]. This makes qualitative evaluation the preferred approach, despite it being subjective, slow, and unscalable. Additionally, the majority of post-hoc analysis methods were proposed and demonstrated for image classification models or for models trained with structured data. This makes it less evident whether those methods would generalise to other domains (e.g., audio, text). This thesis aims to address the above challenges in the context of machine listening models that automatically analyse sounds (e.g., speech, music) using computational models to extract meaningful information.

1.2 Aim

This thesis aims to address some key challenges associated with post-hoc interpretability techniques and focuses on both the design and validation of novel techniques and on their application to analyse the behaviour of machine listening models. The models this thesis uses for experiments aim to perform *singing voice detection* (SVD) that involves detecting the presence of vocals (singing voice) in short-duration musical audio frames (or excerpts) [Humphrey et al., 2019]. The description below highlights some of the key research questions this thesis aims to answer.

Will existing post-hoc interpretability techniques generalise to machine listening models? Section 1.1 mentioned that the majority of post-hoc interpretability techniques were initially demonstrated for the state-of-the-art image classification DNNs (e.g., AlexNet [Krizhevsky et al., 2012]) or for the models trained with structured data [Baehrens et al., 2010, Strumbelj and Kononenko, 2010]. Thus, the application of these methods to machine listening models raises some intriguing questions. For example, will the extension of a method to machine listening require major modifications in the explanation generation pipeline? Are the generated explanations (qualitatively) interpretable, or do they require further processing to extract meaningful information? This thesis analyses these questions by applying three popular post-hoc interpretability techniques to analyse the behaviour of five different SVD models.

Can we design reliable local explanation methods that are better suited to analyse machine listening models? The majority of existing local explanation methods highlight the influence of each feature in a model prediction. Such explanations will be interpretable if the input features are interpretable [Ribeiro et al., 2016a]. However, in machine listening the features for model training (e.g., mel-frequency cepstral coefficients) very often are difficult to interpret in terms of the underlying physical properties of audio signals. The issue of feature interpretability is less severe for machine listening models that use feature learning [Humphrey et al., 2012] as explanations highlight the influence of each spectrogram bin. However, recent research has questioned the reliability of these explanations (see Section 1.1). This thesis aims to address these challenges by asking the following questions. What type of local explanation methods are suitable for machine listening? How can we design and validate the performance of those methods?

What features do the components (layers, neurons) capture in machine listening models trained using feature learning? The analysis of image classification DNNs has demonstrated that those models learn features hierarchically. The results demonstrated that a convolutional neural network (CNN) combines low-level features (e.g., edges, gradients) to construct mid-level features (e.g., textures, contours) and further combines mid-level features to construct high-level features (e.g., object parts) [Zeiler and Fergus, 2014, Nguyen et al., 2016a]. Motivated by these results, this thesis analyses the deep SVD models globally and aims to answer the following questions. Similar to image classification DNNs, do the SVD models also learn high-level class concepts in their output layer neurons? What input features does each layer preserve in a CNN-based SVD model? Can we gain an insight into how a CNN-based SVD model differentiates between the two classification categories (vocals and non-vocals)?

1.3 Thesis structure

- **Chapter 2** provides a literature survey of the two research topics (IML and SVD) that are the focus of this thesis. For IML, the chapter defines key terminologies, discusses the motivations for analysing ML models, provides a detailed survey of different approaches to model interpretability, and describes existing research for analysing machine listening models. For SVD, the chapter defines the use case, highlights its key applications, discusses different approaches to design SVD models, and highlights key research challenges for SVD.

- **Chapter 3** introduces the four singing voice datasets that this thesis uses for the design and analysis of SVD models. Moreover, the chapter describes the five SVD models this thesis uses for experiments. Two SVD models (referred to as *shallow SVD models*) are based on the feature engineering approach [Lehner et al., 2013], and the other three SVD models (referred to as *deep SVD models*) are based on the feature learning approach [Schlüter and Grill, 2015]. The chapter describes the input features, architecture, training methodology, and performance of each SVD model on benchmarked publicly available datasets.

- **Chapter 4** introduces *SoundLIME* (SLIME), a method for the local analysis of any (shallow or deep) machine listening model and demonstrates its effectiveness by using it to analyse the local behaviour of three SVD models (two shallow models and a state-of-the-art deep model). Moreover, the chapter describes experiments that quantitatively analyse the behaviour of the proposed explanation method for different settings of two input parameters. Finally, the chapter introduces a novel technique to generate reliable explanations from the proposed explanation method and demonstrates the technique for a synthetic dataset.

- **Chapter 5** introduces a novel method for analysing the global behaviour of machine learning models. Specifically, the chapter proposes a new approach for *activation maximisation* (AM) and suggests to use a generative adversarial network (GAN) as a prior in the AM pipeline. Moreover, the chapter introduces a novel method to quantitatively select suitable hyperparameter configurations for AM. The chapter describes experiments that demonstrate the proposed methods for two deep SVD models and uses the results to analyse the concepts the output layer neuron(s) capture in both the SVD models. Additionally, the chapter presents a perceptual study to further analyse the observations from the qualitative analysis of synthesised examples corresponding to the two SVD models.

- **Chapter 6** presents *feature inversion*, a method to map DNN features back to the input space and describes the feature inversion approach used in this thesis. The chapter proposes a novel feature inversion-based approach for explaining CNN predictions and demonstrates it by qualitatively analysing the behaviour of a state-of-the-art deep SVD model. Moreover, the chapter quantitatively evaluates the performance of the proposed method. Additionally, the chapter uses feature inversion to visualise and understand the input content preserved by each layer in the SVD model, aiming to gain an insight into the decision-making process of the model.

- **Chapter 7** provides a summary of the key results from the experiments in each chapter and highlights the contributions this thesis makes to interpretable machine learning and machine listening research. Moreover, the chapter presents some ideas to extend this research and provides a general discussion on existing challenges and potential research directions in IML.

1.4 Contributions

The content below mentions the key research contributions of this thesis in each chapter.

Chapter 4

- Three interpretable representations (temporal, spectral, and time-frequency) that assist in extending a local explanation method to machine listening.
- A qualitative and quantitative comparison of SLIME and deconvolutional network-based saliency maps for a state-of-the-art deep SVD model.
- An analysis of the sensitivity of SLIME explanations to two input parameters: the number of perturbed samples in the interpretable space and the content type of the synthetic components.
- Five content types for perturbing input representations (by occlusion) for machine listening models.
- A method to generate reliable local explanations.
- A method to synthesise dataset and ground-truth explanations for the reliability method. The dataset and ground-truth annotations from the experiments are publicly available[1].

Chapter 5

- A method to synthesise examples for maximally (or minimally) activating DNN components (e.g., layers, neurons).
- A method to quantitatively select suitable hyper-parameter values for AM.
- Qualitative understanding about the concepts captured by the output layer neurons in two deep SVD models.

[1] `https://github.com/saum25/local_exp_robustness`

- Qualitative understanding about the interpretability of concepts captured by two types of output neurons (a single sigmoidal neuron and two fully connected neurons) in two deep SVD models.
- A perceptual study that analyses the observations from the qualitative analysis of the synthesised examples corresponding to the two SVD models.

Chapter 6

- A method for explaining predictions of any CNN.
- Qualitative and quantitative evaluation of the proposed local explanation method for a state-of-the-art deep SVD model using two publicly available benchmarked datasets (Jamendo and RWC).
- Evaluation of the normalised reconstruction error at each layer of the deep SVD model for the two publicly available datasets.
- Qualitative understanding about the input information (content) preserved by each layer in the deep SVD model for instances from different datasets.

1.5 Publications

Most of the work presented in this thesis has been presented at international peer-reviewed conferences and workshops or is under review. The list below mentions all the publications related to this thesis.

Peer-reviewed publications associated with this thesis

[1] **Saumitra Mishra**, Bob L. Sturm, and Simon Dixon, "Local Interpretable Model-Agnostic Explanations for Music Content Analysis", in Proceedings of the *18th International Society for Music Information Retrieval Conference (ISMIR)*, pages 537–543, Suzhou, China, October 23–27, 2017.

[2] **Saumitra Mishra**, Bob L. Sturm, and Simon Dixon, "What are You Listening to?" Explaining Predictions of Deep Machine Listening Systems", in Proceedings of the *26th European Signal Processing Conference*, pages 2260–2264, Rome, Italy, September 3–7, 2018.

[3] **Saumitra Mishra**, Bob L. Sturm, and Simon Dixon, "Understanding a Deep Machine Listening Model Through Feature Inversion", in Proceedings of the *19th International Society for Music Information Retrieval*

Conference (ISMIR), pages 755–762, Paris, France, September 23–27, 2018.

[4] **Saumitra Mishra**, Daniel Stoller, Emmanouil Benetos, Bob L. Sturm, and Simon Dixon, "GAN-based Generation and Automatic Selection of Explanations for Neural Networks", in Proceedings of the *International Conference on Learning Representations (ICLR) Workshop on Safe Machine Learning*, New Orleans, USA, May 6–9, 2019.

[5] **Saumitra Mishra**, Emmanouil Benetos, Bob L. Sturm, and Simon Dixon, "Reliable Local Explanations for Machine Listening", in Proceedings of the *International Joint Conference on Neural Networks (IJCNN) Special Session on Explainable Computational/Artificial Intelligence*, Glasgow, Scotland, UK, July 19–24, 2020.

Chapter 4 includes content from [1] and [5]. [1] introduced SLIME, demonstrated its effectiveness for machine listening, compared its performance with saliency maps, and reported preliminary results about the sensitivity of SLIME explanations. [5] performed further analysis about the sensitivity of SLIME explanations to the two input parameters and introduced a method for generating reliable explanations from SLIME. Chapter 5 includes content from [4] that introduced the novel methods to perform AM and hyper-parameter selection and performed preliminary experiments to demonstrate the methods by analysing the concepts captured by the output layer neuron in a deep SVD model. Chapter 6 includes content from [2] and [3]. [2] introduced the feature inversion-based local explanation method and (qualitatively and quantitatively) demonstrated it for a deep SVD model. [3] extended the previous research by first training feature inverters corresponding to each layer of the SVD model and then qualitatively and quantitatively analysing the input content preserved (or lost) at each layer of the SVD model for two benchmarked datasets.

The thesis author is the main contributor to all above mentioned publications, including on the design of experiments, generating and analysing results, and writing and editing papers. Additional contributions from co-authors are described as follows. Daniel Stoller contributed to [4] by sharing a pre-trained GAN, writing the section on the design of the GAN, and reviewing the paper draft. Emmanouil Benetos contributed to papers [4, 5] and Bob L. Sturm and Simon Dixon contributed to all the papers in a supervisory role that involved sharing their opinion on research questions, discussing the experimental results, and reviewing the paper drafts.

Other publications

The thesis author contributed to two more papers (during the same period) presented at international peer-reviewed conferences and workshops. [6] used SLIME to analyse the predictions of a CNN-based replay spoofing detection system. The thesis author contributed by adapting SLIME for the spoofing model, participating in discussions about the intervention experiments, writing the technical description of SLIME, and reviewing the draft. The thesis omits the details of [7] as it does not relate to the research in this thesis.

[6] Bhusan Chettri, **Saumitra Mishra**, Bob L. Sturm, and Emmanouil Benetos, "Analysing the Predictions of a CNN-Based Replay Spoofing Detection System", in Proceedings of the *IEEE Spoken Language Technology Workshop (SLT)*, pages 92–97, Athens, Greece, December 18–21, 2018.

[7] **Saumitra Mishra**, Sreehari Mohan, Khalid Rajab, Gurpreet Dhillon, Pier Lambiase, Ross J. Hunter and Elaine Chew, "Atrial Fibrillation Stratification via Fibrillatory Wave Characterization Using the Filter Diagonalization Method", in Proceedings of the *46th International Conference in Computing in Cardiology (CinC)*, Singapore, September 8–11, 2019.

Chapter 2

Background

As discussed in chapter 1, this thesis mainly involves two research topics - *interpretable machine learning (IML)* and *singing voice detection (SVD)*. This chapter describes both research topics, highlighting state-of-the-art methods and key research challenges for each. Such a description will provide the necessary background for understanding the subsequent chapters of this thesis.

This chapter starts with a brief description of machine learning techniques relevant to this thesis (section 2.1). The next two sections describe research in interpretable machine learning (section 2.2) and singing voice detection (section 2.3), respectively. The section on IML research defines IML, discusses the need for model interpretability, and describes the key methods to analyse machine learning models. The section on SVD research starts by defining SVD and highlighting some of its applications. It later describes common features and popular methods to design SVD models and presents metrics commonly used to evaluate SVD models. The section ends with details about the key SVD research challenges. Finally, this chapter ends with a summary section (section 2.4) that highlights the IML and SVD methods that will be useful in the subsequent chapters of this thesis.

2.1 Machine learning preliminaries

This section briefly describes machine learning techniques used in subsequent chapters of this thesis. Specifically, the section introduces four machine learning models - decision trees and random forests (section 2.1.1), convolutional neural networks (section 2.1.2), and generative adversarial neural networks (section 2.1.3).

2.1.1 Tree-based models

There exist two main types of tree-based models - a single decision tree or a group of decision trees whose predictions are computed using ensemble methods (e.g., bagging [Bishop, 2007]). Tree-based models are popularly used in a supervised setting across classification and regression tasks.

A decision tree is a hierarchical model that learns by recursively partitioning the input space into two or more homogenous subsets using the best features and their best values which the model identifies using some criterion (e.g., minimisation of entropy) [Breiman et al., 1984]. One of the key advantages of decision trees is that they are inherently interpretable and can provide reasons for their predictions. However, explanations generated by decision trees trained with uninterpretable features may sometimes be hard to interpret [Mishra et al., 2017].

A random forest model is an ensemble of decision trees where each tree learns from a random subset of the training dataset and input features [Breiman, 2001]. The methodology of aggregating estimates from multiple decision trees assists in mitigating the problem of high variance associated with the decision tree models. However, the improved performance of random forests comes at the cost of model interpretability.

Chapter 3 describes the design and evaluation of two tree-based models for SVD. This thesis refers to these models as 'shallow SVD models'. Chapter 4 introduces a technique for model analysis and demonstrates it to explain the predictions of the two shallow SVD models.

2.1.2 Convolutional neural networks

A convolutional neural network (CNN) is a feedforward neural network with a specialised architecture to process high-dimensional data (e.g., images) [Goodfellow et al., 2016]. A CNN usually comprises three types of feedforward layers: *convolutional layers* that perform the convolutional operation between an input and a set of learnable filters (kernels), *pooling layers* that reduce the spatial size of an input and *fully-connected layers* that flatten an input to a vector. The use of specialised layers reduces the number of learnable parameters, helping a CNN to scale well to high-dimensional data.

CNNs have been successfully demonstrated for several real-world applications (e.g., image classification, scene segmentation, sentiment classification). In machine listening, CNNs have been used in a wide variety of tasks (e.g., music auto-tagging [Pons et al., 2018], onset detection [Schlüter and Böck, 2014], spoofing detection [Chettri et al., 2018]). Chapter 3 describes an SVD model that uses a nine-layered CNN to achieve state-of-the-art performance on bench-

marked datasets. The thesis refers to this model and its variants as 'deep SVD models' and analyses their behaviour using different post-hoc analysis methods.

2.1.3 Generative adversarial networks

Generative adversarial networks (GANs) are generative models that consist of two DNNs, a generator G and a discriminator D, involved in a minimax two-player game [Goodfellow et al., 2014]. The generator maps a low dimensional noise vector $\mathbf{z}$ to a high dimensional output $G(\mathbf{z};\theta_g)$ (e.g., image). The discriminator is a classifier $D(\mathbf{x};\theta_d)$ that outputs a probability representing that $\mathbf{x}$ came from the data distribution $p_{data}(\mathbf{x})$ and not from the generator. In the GAN framework, the goal of the generator is to synthesise samples that can fool the discriminator and the goal of the discriminator is to be successfully able to identify whether an input is real or fake. It is expected that after successful training the samples G generates are close to those sampled from the real data distribution, making it extremely difficult for D to differentiate between real and synthetic samples.

Goodfellow et al. [2014] proposed to train both the models simultaneously using the backpropagation algorithm. Formally, D and G are involved in a two-player minimax game with value function $V(G,D)$ given as

$$\mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log(D(\mathbf{x}))] + E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))]. \tag{2.1}$$

The discriminator and generator try to maximise and minimise the above value function, respectively. Importantly, in the early stages of GAN training, optimising the above loss function may result in a vanishing gradient problem for the generator model. In such a scenario Goodfellow et al. [2014] suggested to train G to maximise $\log(D(G(z)))$.

Another way to address the problem of vanishing gradients is by using an alternate loss function called Wasserstein distance (or the earth mover distance). GANs trained using Wasserstein distance are termed as Wasserstein GANs (WGANs) [Arjovsky et al., 2017]. In WGANs, the discriminator is termed as 'critic' and it does not classify an input sample into the real and fake categories. Instead, the critic outputs a scalar that may or may not be less than 1 or greater than 0 and aims to make the output bigger for the real samples than for the fake samples. The generator and critic aim to maximise the functions $D(G(\mathbf{z}))$ and $D(\mathbf{x}) - D(G(\mathbf{z}))$, respectively. Additionally, for the computation of Wasserstein distance, the critic model D has to be a 1-Lipschitz function. WGAN imposes this constraint on D by clipping its weights to fall within a range controlled by a hyper-parameter. Formally, if $\mathbf{w}_D$ represents the weights of the critic model, WGAN performs clipping by using a hyper-parameter c to

perform $\mathbf{w}_D \leftarrow clip(\mathbf{w}_D, -c, c)$.

The WGAN method assists in improving the training stability of GANs. However, in WGAN, the performance of the critic model is very sensitive to the hyper-parameter c. Moreover, weight clipping acts as a regulariser, restricting the capacity of the critic model. Hence, another approach called WGAN-GP [Gulrajani et al., 2017] proposed to impose the Lipschitz constraint by applying a gradient penalty. WGAN-GP penalises the critic model if the gradient norm of its output with respect to input deviates from 1. Formally, the critic loss function is given as

$$L_D = \mathbb{E}_{\tilde{\mathbf{x}} \sim p_g}[D(\tilde{\mathbf{x}})] - \mathbb{E}_{\mathbf{x} \sim p_{data}}[D(\mathbf{x})] + \lambda \cdot \mathbb{E}_{\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}}(||\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})||_2 - 1)^2 \qquad (2.2)$$

where $\tilde{\mathbf{x}} = G(\mathbf{z})$ for $\mathbf{z} \sim p_{\mathbf{z}}$ and $\hat{\mathbf{x}} = \beta \cdot \tilde{\mathbf{x}} + (1 - \beta) \cdot \mathbf{x}$ for $\beta \in [0, 1]$. The experiments in Chapter 5 use a GAN trained using the WGAN-GP framework to understand the features that the output layer neurons in two deep SVD models have learnt to identify.

In recent years, several variants of the GAN model proposed by Goodfellow et al. [2014] have been developed to improve the quality of generated samples and stabilise GAN training. As subjective evaluations can be challenging and expensive, several quantitative measures have been proposed to evaluate and compare the performance of GANs. Fréchet Inception Distance (FID) [Heusel et al., 2017] is one such metric that quantifies the quality of GANs. FID compares the distributions of activations from real and synthetic examples (e.g., images). The activations are the output of the pool3 layer in Inception-v3, an image classification model [Szegedy et al., 2016]. The FID $d(.,.)$ is defined as the distance between two multidimensional Gaussian distributions of activations from the Inception-v3 model and is given as

$$d^2((\mathbf{m}_r, \mathbf{C}_r), (\mathbf{m}_s, \mathbf{C}_s)) = ||\mathbf{m}_r - \mathbf{m}_s||_2^2 + \mathrm{Tr}\Big(\mathbf{C}_r + \mathbf{C}_s - 2(\mathbf{C}_r\mathbf{C}_s)^{\frac{1}{2}}\Big), \qquad (2.3)$$

where ($\mathbf{m}_r$, $\mathbf{C}_r$) and ($\mathbf{m}_s$, $\mathbf{C}_s$) are the mean and covariance corresponding to the activation distribution of the real and synthetic samples, respectively and Tr is the trace operator. A lower value of FID is desired and it suggests that the activation distribution of the synthetic samples is similar to the real samples. In Chapter 5, the FID is used to quantitatively select the best hyper-parameters for performing activation-maximisation to understand a DNN.

2.2 Interpretable machine learning

In recent years, there has been a lot of interest in analysing the behaviour of machine learning models. Researchers have designed numerous methods and demonstrated them for bringing interpretability to different kinds of machine learning models for a multitude of applications [Montavon et al., 2018, Du et al., 2018a, Gilpin et al., 2018, Guidotti et al., 2019, Murdoch et al., 2019].

This section provides a detailed survey of the IML research field by defining key terminologies (section 2.2.1), discussing the need for model interpretability (section 2.2.2), describing popular IML methods (section 2.2.3), and highlighting existing research in understanding machine listening models (section 2.2.4).

2.2.1 What is interpretability?

Although numerous papers exist claiming to bring *interpretability* to machine learning models, there is a lack of agreement about what interpretability means in the context of machine learning [Lipton, 2016]. For instance, Miller [2019] defines interpretability as "the degree to which an observer can understand the cause of a decision", which refers to analysing why a machine learning (ML) model takes a particular decision. On the other hand, Kim et al. [2016] define interpretability as "the degree to which a user can correctly and efficiently predict a model's decisions", which refers to analysing model's behaviour for multiple instances to gain an insight into factors influencing its predictions. We can gain such an insight even without analysing a model internally, especially if the model predicts by exploiting confounders in the dataset [Chettri et al., 2018]. Moreover, in a different direction, [Rudin, 2019] argues that interpretability is domain-dependent and thus providing a single definition for it may not be appropriate.

This thesis follows the definition from Doshi-Velez and Kim [2017] that defines "interpretability as the ability to explain or present [the behaviour of a machine learning model] in understandable terms to a human", which refers to analysing both why and how a model predicts. Machine learning models may inherently possess such an ability (this thesis refers to such models as *interpretable models*, e.g., decision trees) or may require auxiliary methods (e.g., saliency maps [Simonyan et al., 2014]) to acquire it[1]. Moreover, the authors also noted that the constituents of an explanation are also not formalised. Murdoch et al. [2019] suggested that an explanation may take different forms depending on the end-user.

[1] In literature, researchers refer to such machine learning models as 'black-box' models (e.g., deep convolutional neural networks) [Rudin, 2019].

Using the above definitions, this thesis defines IML as the field of ML research that involves designing and applying interpretable machine learning models and post-hoc methods to make black-box models interpretable. Section 2.2.3 provides more details about these methods.

In the literature, instead of interpretability, some researchers use the term *explainability*, and instead of IML, some researchers use *explainable ML or explainable AI (XAI)*. Moreover, in some works, researchers use the term *transparent models* instead of interpretable models. There is again no agreement in the research community on the exact meanings of these terms, and researchers frequently use them interchangeably [Lipton, 2016, Gilpin et al., 2018, Du et al., 2018a, Molnar, 2019]. This thesis only uses the terms interpretability, IML, and interpretable models using the definitions above.

There are some recent efforts in distinguishing IML and other related terms, but the proposed definitions lack consistency. For instance, Rudin [2019] argues that IML deals with training inherently interpretable models, while explainable machine learning involves explaining black-boxes using post-hoc methods. On the other hand, Gilpin et al. [2018] argue that gaining some insight into a model's working is interpretability, while explainability is more complex that involves analysing multiple aspects of a machine learning model. They mention that "explainable models are interpretable by default, but the reverse is not always true."

2.2.2 Need for model interpretability

The need for interpretability in machine learning arises when a machine learning model needs to satisfy auxiliary criteria (e.g., robustness) in addition to optimising pre-defined quantifiable metrics (e.g., classification error). The majority of auxiliary criteria are not quantifiable[2], creating an incompleteness in the problem specification [Doshi-Velez and Kim, 2017]. For instance, in addition to having high accuracy on a test dataset, an ML model might need to be trustworthy (an auxiliary criterion), but mapping trust to a function for optimisation is an ambiguous task. One may argue that high accuracy on a test dataset indicates that the model is trustworthy. However, ML models are capable of reporting high performance by exploiting artefacts and confounders in datasets [Montavon et al., 2018, Samek et al., 2019]. In literature, researchers refer such models as "Horses" [Sturm, 2014] and "Clever Hans predictors" [Lapuschkin et al., 2019]. In such a scenario, interpretability helps to qualitatively verify if the model is trustworthy by explaining the reasoning behind its predictions that a user can

[2]Recent research in fairness and privacy in the context of machine learning has formalised their meanings [Doshi-Velez and Kim, 2017].

verify using domain knowledge.

Below we mention some auxiliary criteria that ML models may need to satisfy and highlight how interpretability may help in fulfilling them. For more auxiliary criteria, refer to Lipton [2016], Doshi-Velez and Kim [2017].

- Trust: In safety-critical applications (e.g., healthcare), it is essential that humans trust the ML models they are deploying (or using). One of the ways to gain trust is by using interpretability to verify that dataset faults or confounders do not influence the predictions of an ML model [Guidotti et al., 2019, Du et al., 2018a].

- Fairness: Recent research has shown that ML datasets have inherent biases, and it is challenging to prevent ML models trained using biased datasets from being discriminatory [Gilpin et al., 2018]. We can prevent the use of some features in model training, but it is hard to pre-identify implicit correlations present in datasets. For instance, Angwin et al. [2016] show that a tool[3] used to identify the risk of a criminal reoffending was racially biased. Interpretability may help to highlight unfair models by analysing if a model uses discriminatory features for its predictions.

- Robustness: ML models are non-robust in the sense that their predictions are sensitive to imperceptible adversarial perturbations [Szegedy et al., 2014]. Researchers have proposed numerous methods to defend ML models against adversarial inputs, but none of them guarantees a perfectly robust ML model. The lack of robustness in ML models is a serious concern when we deploy them in safety-critical applications (e.g., self-driving cars). However, we argue that using interpretability for analysing adversarial examples would provide further understanding of their characteristics and behaviour which may assist in developing robust ML models. For example, visualising the saliency-map-based local explanations (see Section 2.2.3.2) for DNN predictions corresponding to an input and its adversarial version would highlight input features that assist the DNN in differentiating between the two inputs. Recently, Noack et al. [2019] have demonstrated that constraining image classification DNNs to have interpretable gradients makes them comparatively more robust to adversarial examples than DNNs trained in a standard manner.

- Legal requirements: A recent European Union regulation, General Data Protection Regulation (GDPR), now provides users of automated decision-making systems a right to explanation [Goodman and Flaxman, 2017,

[3]Correctional Offender Management Profiling for Alternative Sections (COMPAS)

Mittelstadt et al., 2019]. Interpretability techniques will help in explaining predictions of ML models in understandable terms to users.

In addition to handling the problem of incompleteness in specifications, interpretability helps in debugging ML models (e.g., by analysing inputs for which a model fails to predict accurately [Du et al., 2018a]) and in extracting new knowledge (e.g., scientific insights, game-playing strategies) previously unknown to humans [Montavon et al., 2018, Samek et al., 2019].

Interpretability also helps to improve ML datasets by identifying if an ML model is utilising dataset artefacts for its predictions. For instance, Ribeiro et al. [2016b] discovered that a deep neural network (DNN) was discriminating between huskies and wolves by analysing the presence or absence of ice in input images. One can easily improve such a dataset by adding images containing the animals in non-snow regions. Similarly, interpretability helps in improving deep learning architectures. For instance, Zeiler and Fergus [2014] visualised features from the first two convolutional layers of a state-of-the-art convolutional neural network (CNN) to discover that larger filter and stride sizes result in poor learning. They used this insight and reduced the filter and stride sizes resulting in a then-state-of-the-art image classification model.

2.2.3 Methods for model interpretability

This section discusses methods that aim to analyse the behaviour of machine learning models. IML is a rapidly developing field, and researchers have proposed numerous methods that claim to bring interpretability to ML models. Recently, there has been some initial work in creating a taxonomy to group existing IML methods [Guidotti et al., 2019, Gilpin et al., 2018]. Although there is a lack of consistency in the proposed taxonomies, broadly, we can categorise IML methods into two major categories: the first one involves methods that design interpretable models (e.g., decision trees); and the second one involves methods that perform post-hoc analysis of trained 'black-box' ML models (e.g., CNNs). The subsequent sections discuss some methods from each category and their corresponding sub-categories. We have tried to select a good number of methods from each category (and sub-category) to provide a detailed overview of research directions within each category (and sub-category).

2.2.3.1 Methods for designing interpretable models

This category of methods aims to design models that are inherently interpretable and do not need post-hoc methods to explain their behaviour. Such models usually require some constraints during their training and these constraints may

take different forms (e.g., sparsity, monotonicity, case-based reasoning) depending on the problem domain [Rudin, 2019]. In literature, some researchers refer to these models by other names, e.g., transparent boxes [Guidotti et al., 2019], explanation-producing systems [Gilpin et al., 2018], intrinsically interpretable models [Du et al., 2018a]. We can analyse the behaviour of these models by directly inspecting model components (e.g., a path in a decision tree) [Ribeiro et al., 2016a]. This section presents methods that can be clustered in some coherent theme.

One category of methods trains *rule-based (or logical)* models that are composed of a set of (dependent or mutually-exclusive) rules. The most well-known representative of such models is the decision tree model that consists of if-else rules [Breiman et al., 1984]. Some recent works have proposed decision lists that are less complex and more accurate extensions of the decision tree models (e.g., falling rule lists [Wang and Rudin, 2015], Bayesian rule lists [Letham et al., 2015]). These models are composed of an ordered sequence of if-then-else statements, and they take a decision whenever a rule is true. Lakkaraju et al. [2016] proposed interpretable decision sets that extend decision lists by making the if-then rules independent, resulting in more interpretable models. Finally, Wang et al. [2016] also extended decision lists and proposed rule sets that consist of a small set of short rules, where each rule is a conjunction of conditions.

The second category of methods designs interpretable models using *prototype selection and case-based reasoning*. The definition of a prototype is application-specific. For instance, a prototype can be an average of the instances belonging to one of the classification categories in the training dataset. These models predict by measuring the similarity of a test instance with each element in the prototype set. Kim et al. [2014] trained an interpretable Bayesian case model (BCM) by first performing unsupervised clustering and then learning the prototypes (one per cluster) and subspaces. Each subspace is a subset of features characterising a prototype. The learning of low-dimensional subspaces relates to encouraging sparsity. The learnt prototypes and subspaces provide interpretability and assist in generating explanations during inference. In subsequent work, [Kim et al., 2016] extended BCM to learn criticisms that are counter-examples for each cluster.

The third category of methods uses *special architectures or training methodologies* to make 'black-box' deep learning models interpretable. Arguably, the most popular way to bring some interpretability to DNNs is to use attention mechanisms that provide a weighting over the input or feature space to help visualise input sections a model attends to while performing a prediction [Xu et al., 2015]. Another method used a special prototype layer to train interpretable DNNs for different computer vision applications [Li et al., 2018, Chen

et al., 2019]. The prototype layer generates prototypes (e.g., parts of an image) for each classification category and uses them during inference for case-based reasoning. The performance of these models was comparable to complex, uninterpretable state-of-the-art models. Zhang et al. [2018] proposed interpretable CNNs by adding a regularisation loss at the deepest convolutional layer that forces the model to learn disentangled representations resulting in filters that capture semantically meaningful features. Alvarez-Melis and Jaakkola [2018] proposed self-explaining neural networks consisting of three components: a deep encoder, a parametrizer that generates relevance scores and an aggregation function that combines scores to generate predictions.

There exist some methods that do not fall under the previously defined categories. For instance, one approach trained interpretable models by adding a sparsity constraint that encourages a model to use fewer features during inference [Ustun and Rudin, 2015]. These models generally work well for structured data. Another approach applied generalised additive models with pairwise interactions for healthcare applications and demonstrated that these interpretable models perform comparably to state-of-the-art black-box models [Caruana et al., 2015]. Table 2.1 summarises all the methods and corresponding models we have discussed so far.

Although interpretable models provide some insight into their behaviour, avoiding the need for post-hoc methods, these models face multiple challenges [Ribeiro et al., 2016a]. For instance, if the input features used for model training are uninterpretable, the explanations (in terms of input features) will also be uninterpretable [Mishra et al., 2017]. Moreover, there is a trade-off between model flexibility and model interpretability [Freitas, 2013]. Generally, interpretable models have limited expressive power (complexity of the learnt functions), resulting in poor performance on tasks that use high-dimensional data. State-of-the-art models for a vast majority of complex machine learning tasks in multiple domains (e.g., vision, language, audio) are uninterpretable DNNs [4]. These challenges have motivated the use of post-hoc methods to explain machine learning models that the next section discusses in detail.

2.2.3.2 Methods for post-hoc interpretability

Post-hoc interpretability involves analysing the behaviour of pre-trained ML models using specialised methods [Lipton, 2016, Ribeiro et al., 2016a]. This category of model analysis does not put any constraints on ML models during the training phase. Thus, the methods are useful to analyse the behaviour of any

[4]Some recent works have demonstrated that using specialised architectures and training methods we can train interpretable versions of DNNs that perform comparably to state-of-the-art methods [Rudin, 2019].

Methods	Models	References
Rule-based	Decision trees	Breiman et al. [1984]
	Rule lists	Wang and Rudin [2015], Letham et al. [2015]
	Decision sets	Lakkaraju et al. [2016]
	Rule sets	Wang et al. [2016]
Prototype selection and case-based reasoning	Bayesian case model	Kim et al. [2014], Kim et al. [2016]
Specialised architectures	Attention-based models	Xu et al. [2015]
	Case-based reasoning models	Li et al. [2018] Chen et al. [2019]
	Interpretable CNNs	Zhang et al. [2018]
	Self-explaining neural networks	Alvarez-Melis and Jaakkola [2018]
Miscellaneous	Sparse linear models	Ustun and Rudin [2015]
	Generalised additive models	Caruana et al. [2015]

Table 2.1: Some methods to train interpretable models.

black-box ML model (e.g., a neural network) without sacrificing its predictive capacity. We can categorise methods for post-hoc interpretability into two main categories: methods that analyse ML models *globally*, and methods that analyse ML models *locally* [Montavon et al., 2018]. Moreover, the methods within each category can be model-agnostic (applicable to any ML model) or model-specific. In the following sections, we discuss some leading post-hoc analysis methods.

Methods to analyse ML models globally

The methods from this category aim to understand the global behaviour of any pre-trained ML model. The global behaviour of an ML model refers to characteristics that are consistent across input instances. For instance, visualisation of features (latent representations) from an image classification CNN reveals that the neurons in the first convolutional layer function as edge detectors, capturing edges in different orientations [Zeiler and Fergus, 2014]. This characteristic of the image classification model is consistent across input images, providing an insight into the global behaviour of the model.

Researchers have proposed multiple methods for global analysis that depend on the ML model and the characteristics we want to analyse. For instance, we can analyse the global behaviour of ML models that use hand-crafted features by understanding the influence of each input feature on model performance. One way to do this uses *permutation feature importance*, a method that assigns importance scores to input features by permuting each feature value across a test dataset, and recording the change in some measure of model performance (e.g., classification error). The feature whose value permutations maximally change the model performance measure has maximum influence on model predictions [Du et al., 2018a, Molnar, 2019]. This method may help analyse simple ML models[5], but does not scale well to models trained using high-dimensional data. Moreover, even for simple ML models, this method does not capture correlations among features, and if we consider them (e.g., by permuting values of pairs of features), this method becomes unscalable.

One way to analyse the global behaviour of complex ML models is by using interpretable models (e.g., decision trees) to approximate the global behaviour of complex ML models. In literature, researchers refer to such interpretable models as *interpretable proxies* or *surrogate models*. Some early efforts from this category proposed interpretable proxies for neural network models. For instance, Thrun [1994] used validity interval analysis, a method to analyse the input-output behaviour of neural networks, to extract if-then rules from a trained neural network. Similarly, Craven and Shavlik [1995] proposed an algorithm

[5]Models that use low dimensional features as inputs.

that queries a neural network model to construct a decision tree that represents the concept encoded by the model. Recently, Carmona et al. [2015] extended the above approaches to interpret a matrix factorisation model using two proxies: a Bayesian network and simple logic rules. The interpretable proxies may provide some useful insight into the behaviour of a complex model, but in some cases, those insights may be unfaithful due to the oversimplification of the complex model. Moreover, the proxy models may themselves become fairly complex (e.g., very deep decision trees), and hinder interpretability [Gilpin et al., 2018].

In another direction, there exist methods specific to the ML models we want to analyse. For instance, researchers have proposed methods to understand DNNs by analysing their latent representations. These representations refer to the features that different components (e.g., neurons, layers) of a DNN have learnt to identify. There are two main ways to perform such an analysis: synthesising examples to maximally activate DNN components (*activation maximisation*), or mapping latent features back to the input space (*feature inversion*). The following sections discuss both the methods in detail.

Activation maximisation Research in neuroscience has demonstrated the presence of multifaceted "grandmother" cells in the human brain that fire (get maximally activated) for high-level semantic concepts (e.g., celebrity names) [Quiroga et al., 2005]. This discovery motivated researchers working on the analysis of DNNs to assume that similar to the grandmother cells, higher layer neurons in a DNN capture disentangled representations (semantically meaningful concepts) [Goodfellow et al., 2009, Erhan et al., 2009, Le et al., 2012]. To understand these concepts, researchers used activation maximisation (AM), a method that synthesises examples in the input space (e.g., images) to maximally activate neurons in a DNN [Erhan et al., 2009]. This thesis refers to such AM as *vanilla AM*, to differentiate it from its variant that identifies dataset instances that maximally activate DNN neurons. This thesis refers to this variant as *data-driven AM* that analyses a DNN in three steps.

- First, select N_{ij}, the i-th neuron (unit) in the j-th layer of the DNN. We want to understand features that maximally activate N_{ij}.
- Then, feed input data (e.g., from the training dataset) to the DNN and record the activation set A_{ij} whose elements a_{ij}^k represent the activation (output) of N_{ij} for the k-th input.
- Finally, sort A_{ij} and select the top M activations and their corresponding inputs. Assume that the top M inputs contain some common features (concepts) that maximally activate N_{ij}. To further investigate the features, we can

- visually analyse the top-M inputs to identify common concepts [Girshick et al., 2014]. One can reduce the task complexity by using the receptive field of N_{ij} to restrict the look-up area within each of the top-M inputs [Zhou et al., 2015].
- segment each of the top-M inputs into semantically meaningful components and iteratively remove components to analyse the change in neuron activation [Zhou et al., 2015].

Researchers have applied data-driven AM to analyse CNNs trained for computer vision applications (e.g., image classification, object detection) [Girshick et al., 2014, Agrawal et al., 2014]. They discovered that deeper layer neurons (e.g., from the deepest max-pooling layer) get maximally activated for high-level semantic concepts (e.g., people, dog faces, buildings), a behaviour that is similar to the grandmother cells in the human brain. However, they also noted that not all neurons learn individual concepts, and there were multiple instances where random linear combinations of neurons also get maximally activated by high-level concepts in input examples [Szegedy et al., 2014], suggesting that deeper DNN layers learn both individual and distributed concepts [Zhou et al., 2015].

Data-driven AM may assist in gaining some insights into the global behaviour of a DNN, but there are multiple challenges [Nguyen et al., 2016a]. For instance, this method requires knowledge about the data distribution used to train the model, which may not be available in some scenarios. Moreover, often high-dimensional inputs consist of multiple components (e.g., objects in an image) and using dataset-based AM generally highlights multiple components, making the identification of semantic concepts challenging. For example, Nguyen et al. [2016a] noted that if a neuron is maximally activated for an image of a lawn mower on green grass, this method does not clarify if the neuron is sensitive to the lawn mower or the green grass or both (context). Due to these challenges, vanilla AM has often been a better approach to analyse the global behaviour of DNNs as opposed to data-driven AM as the examples vanilla AM synthesises will only include features that maximally influence DNN components.

Vanilla activation maximisation synthesises patterns in the input space (e.g., images) to maximally activate a neuron in a DNN [Erhan et al., 2009, Simonyan et al., 2014]. The vanilla AM algorithm considers the task of preferred stimulus synthesis as an optimisation problem. Thus, for a trained model (e.g., a CNN), vanilla AM synthesizes an input $\mathbf{x}_{ij}$ that maximises the activation $a_{ij}(\mathbf{x}_{ij})$ of N_{ij} (the i_{th} neuron in the j_{th} layer). To do this, the algorithm starts at some input $\mathbf{x} = \mathbf{x}_0$, and then moves in the input space along the direction of the gradient $\partial a_{ij}(\mathbf{x})/\partial \mathbf{x}$ to generate an input $\mathbf{x}_{ij}$ that maximises the activation value for N_{ij}. $\mathbf{x}_0$ can take multiple forms. For instance, it can be a random input

or can be an average of all examples in the training dataset. Researchers have demonstrated that performing activation maximisation in this way very often synthesises examples that do not resemble those from the natural data distribution. For instance, images synthesized using such an unconstrained optimisation process have unreal properties (e.g., high pixel values, high-frequency patterns, and no global structure) [Nguyen et al., 2015]. To avoid producing such unrealistic inputs, researchers proposed to regularise the objective function of the vanilla AM algorithm [Yosinski et al., 2015]. Formally, the regularised vanilla AM optimises

$$\mathbf{x}_{ij} = \arg\max_{\mathbf{x}}(a_{ij}(\mathbf{x}) - r_{\alpha}(\mathbf{x})) \tag{2.4}$$

where $r_{\alpha}(\mathbf{x})$ is a parameterised regularisation function (prior) that aims to keep the synthesised example ($\mathbf{x}_{ij}$) realistic.

Researchers have proposed several hand-crafted regularisers and demonstrated that iteratively regularising inputs with suitable regularisers helps vanilla AM to synthesise high-quality examples [Nguyen et al., 2016b, Wei et al., 2015, Olah et al., 2017]. Some of these regularisers are useful for any data type (e.g., L_2 norm (Simonyan et al. [2014])), while some regularisers implicitly assume the input to be an image (e.g., Gaussian blur[6] ([Yosinski et al., 2015]), total variation ([Mahendran and Vedaldi, 2015])).

Hand-crafting regularisers is complex and gets increasingly difficult for high-dimensional data (e.g., images, audio) where it is ambiguous to define the properties that separate a realistic input from an unreal one. Thus, to avoid this complexity, Nguyen et al. [2016a] proposed to replace hand-crafted priors by the learned priors in vanilla AM and to perform the optimisation in the latent space of the learned prior. The authors used deep generator networks from Dosovitskiy and Brox [2016b] as learned priors. The input to these generators is a d-dimensional latent code that they map to the most-likely synthetic image. The authors trained the generators using images from the ImageNet training dataset [Russakovsky et al., 2015] and the training objective comprised of three loss functions: L_2 loss in the input space, L_2 loss in the feature space, and the adversarial loss [Goodfellow et al., 2014]. Formally, if $\Omega \in \mathbb{R}^d$ represents the latent code, $G(\Omega)$ represents the generator output, and a_{ij} represents the activation function of the neuron N_{ij} (the i_{th} neuron at j_{th} layer) we want to analyse, then the learned prior-based vanilla AM optimises

$$\Omega_{ij} = \arg\max_{\Omega}(a_{ij}(G(\Omega)) - \lambda \cdot \|\Omega\|_2) \tag{2.5}$$

[6]The Gaussian blur regulariser blurs the modified input at each vanilla AM iteration, reducing the effect of structured high-frequency components generated via gradient ascent.

where λ is a scaling constant. Finally, to generate the preferred example $\mathbf{x}_{ij}$ for the neuron N_{ij}, the method performs a forward pass through the generator using the optimised latent code Ω_{ij} from Eq. 2.5 , i.e., $\mathbf{x}_{ij} = G(\Omega_{ij})$. Nguyen et al. [2016a] reported high-quality visualisations using the learned prior-based vanilla AM. Their analysis of the state-of-the-art image classification models revealed that the models learn individual class concepts in the output layer neurons. Moreover, their experiments also confirmed a previous observation that the neurons in image classification models learn features from input images hierarchically. The lower layer neurons learn low-level features (e.g., vertical/horizontal lines, textures), middle layer neurons seem to combine low-level features to learn mid-level features (e.g., object parts), and finally, the deeper layer neurons learn high-level class concepts (e.g., a dog) [Zeiler and Fergus, 2014]. Chapter 5 describes experiments that use both the data-driven and vanilla versions of AM to analyse the behaviour of a machine listening model.

We can also apply vanilla AM to understand features that a group of neurons or layers in a DNN are sensitive to [Szegedy et al., 2014, Agrawal et al., 2014, Yosinski et al., 2014, Mordvintsev et al., 2015]. To do this, vanilla AM generates examples to maximise the sum of activations of a group of neurons or the sum of all the activations in a layer. Additionally, we can use vanilla AM to understand the features that a DNN is invariant to by first randomly sampling multiple initial inputs to the algorithm (e.g., an input image, a latent code) and then for each initial input synthesising the maximally activating example for further analysis. For instance, Yosinski et al. [2015] demonstrated that their image classification CNN was invariant to the orientation of image objects.

Although vanilla AM has been very useful in analysing the global behaviour of DNNs, sometimes the examples it synthesises for understanding the deeper layer neurons can be difficult to interpret. This happens as those neurons are multi-faceted which means they get maximally activated for different representations of the same feature [Nguyen et al., 2016b, Montavon et al., 2018]. For example, a face detecting neuron will maximally activate for a human as well as an animal face. Vanilla activation maximisation can generate many prototypes that can maximally activate the face detecting neuron, but as the algorithm aims at finding a single solution, the synthesised example can sometimes become too complex to interpret.

Feature inversion Another way to analyse the global behaviour of a DNN is by using *feature inversion* that maps (inverts) DNN features[7] at any layer back to the input space (e.g., an image) [Mahendran and Vedaldi, 2015, Dosovitskiy and Brox, 2016a]. Inverting features from each layer of a DNN trained for

[7]Features refers to the outputs of DNN layers.

input classification will highlight the information that the model preserves at each layer, providing an insight into the input features that the DNN uses for predictions (see Chapter 6). Additionally, although this thesis describes feature inversion in the context of DNNs, it is a generic method that one can use to invert hand-crafted features [Dosovitskiy and Brox, 2016a].

There exist two key methods to perform feature inversion for a DNN. The first method generates an inverted representation $\hat{\mathbf{x}}_L \in \mathbb{R}^n$ from an L^{th} layer feature by iteratively minimising the feature space loss between an input $\mathbf{x} \in \mathbb{R}^n$ (e.g., image) and an intermediate representation $\mathbf{x}'_L \in \mathbb{R}^n$ [Mahendran and Vedaldi, 2015]. The method starts with a randomly sampled $\mathbf{x}'_L$ and in each iteration updates it by calculating the gradient of the loss function at $\mathbf{x}'_L$. Formally, given a DNN with representation function $\Phi_L : \mathbb{R}^n \rightarrow \mathbb{R}^d$ that maps an n-dimensional input to a d-dimensional feature $\Phi_L(\mathbf{x})$ at a layer L, the method inverts $\Phi_L(\mathbf{x})$ by solving

$$\hat{\mathbf{x}}_L = \underset{\mathbf{x}'_L}{\arg\min} \|\Phi_L(\mathbf{x}'_L) - \Phi_L(\mathbf{x})\|_2^2 + \beta \cdot \Omega(\mathbf{x}'_L) \tag{2.6}$$

where $\Omega : \mathbb{R}^n \rightarrow \mathbb{R}$ is a regularisation function and β is a scaling constant. The regularisation function constrains the optimisation (imposes a data prior) and limits the search to a valid data manifold. This is crucial as previous research has shown that unrestricted search may output fooling examples [Nguyen et al., 2015].

Mahendran and Vedaldi [2015] applied the above method to analyse the behaviour of AlexNet, a CNN for image recognition. AlexNet was the winner of the classification and detection task in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2012 [Krizhevsky et al., 2012]. In order to impose a natural image prior, they proposed two hand-crafted regularisers: α-norm and total variation norm. Their work demonstrated that the inverted features from the deepest convolutional layer in AlexNet are visually similar (preserve the spatial layout and colour) to the input image. They also demonstrated that although the reconstructions from fully connected layers are visually poor, they still depict the presence of high-level features (e.g., the facial features of an animal). Their work also highlighted the invariances captured by the AlexNet layers. For example, the inverted representations from the deepest layer in the model (a fully connected layer) depicted an object at different locations, orientations and scales.

Instead of beginning from a randomly sampled $\mathbf{x}'_L$, the above method can use a more structured starting input (e.g., medoids of clusters corresponding to classification categories). This would make the optimisation task comparatively less challenging and may generate interpretable visualisations even with weak

priors (e.g., L_2 norm). For example, Du et al. [2018b] extended the method by Mahendran and Vedaldi [2015] by replacing the randomly sampled $\mathbf{x}'_L$ in Eq. 2.6 by $\tilde{\mathbf{x}}_L \in \mathbb{R}^n$, a weighted sum of $\mathbf{x}$ and a background representation $\mathbf{p} \in \mathbb{R}^n$. They define $\tilde{\mathbf{x}}_L$ as

$$\tilde{\mathbf{x}}_L = \mathbf{x} \odot \mathbf{m} + \mathbf{p} \odot (1 - \mathbf{m}) \tag{2.7}$$

where $\odot$ represents the element-wise multiplication, $\mathbf{p}$ is a perturbed (e.g., blurred) version of the input $\mathbf{x}$ and $\mathbf{m} \in [0, 1]^n$ is a weight vector indicating the significance of each input dimension in generating the feature $\Phi_L(\mathbf{x})$. They also propose a new hand-crafted regulariser that uses the weighted sum of feature maps from intermediate CNN layers.

The methods by Mahendran and Vedaldi [2015] and Du et al. [2018b] are helpful in understanding features of a DNN. However, they have two limitations: (1) hand-crafting a prior is challenging as it is hard to define the composition of a naturally occurring input (e.g., image, audio); and (2) the method needs to solve Eq. 2.6 for every new feature it needs to invert.

Dosovitskiy and Brox [2016a] proposed the second feature inversion method that tackles both the above issues and generates visually improved reconstructions even for the fully connected layers of AlexNet. The method trains a separate DNN (this thesis refers to it as a "feature inverter") to invert the features from each layer of AlexNet. The method uses an up-convolutional neural network [Dosovitskiy et al., 2015] as a feature inverter. The method trains a feature inverter by minimising the input space loss Ψ_{input}, defined as the squared Euclidean distance between an input image and its inverted representation. Although this method learns a natural image prior implicitly during training and is expensive only at the training time,[8] the inverted representations are blurry for all the layers. The reason behind this is the way a feature inverter inverts a feature. A forward pass through AlexNet (or any DNN) maps several inputs to the same feature. Thus, to invert a feature, a feature inverter generates an input that is an average of all the inputs that AlexNet maps to the given feature. This averaging effect results in blurry reconstructions. We can think of a feature inverter as an approximate inverse of the representation function Φ_L. Φ_L is a many-to-one function, as several inputs can have the same feature representation. This prevents Φ_L from having a unique inverse.

In addition to works that propose new methods for feature inversion, there has been some work in applying feature inversion for high-quality image generation and for understanding state-of-the-art CNN models. Gatys et al. [2016] used feature inversion to extract content information from an input image and

[8] We need to train one feature inverter per layer of a DNN. Once feature inverters are trained, feature inversion happens in near real time.

Categories	Methods	References
Feature importance	Permutation feature importance	Du et al. [2018a], Molnar [2019]
Interpretable proxies	If-then rules	Thrun [1994], Carmona et al. [2015]
	Decision trees	Craven and Shavlik [1995]
	Bayesian networks	Carmona et al. [2015]
Model specific	Vanilla activation maximisation	Erhan et al. [2009], Simonyan et al. [2014]
	Data-driven activation maximisation	Girshick et al. [2014], Zhou et al. [2015]
	Feature inversion	Mahendran and Vedaldi [2015], Dosovitskiy and Brox [2016a]

Table 2.2: Some methods to analysis the global behaviour of ML models.

then combine this information with style information from another image to perform neural style transfer. Upchurch et al. [2017] performed high-level semantic transformations to an input image by first linearly interpolating deeper layer feature vectors from a CNN and then by inverting the interpolated features back to the image space. As discussed above, Mahendran and Vedaldi [2015] and Dosovitskiy and Brox [2016a] applied feature inversion to understand AlexNet. For example, Dosovitskiy and Brox [2016a] discovered that AlexNet preserves the colour and approximate location of an object in its last fully connected layer. Later, Mahendran and Vedaldi [2016] extended the previous work by applying feature inversion to understand variants of the VGGNet model [Simonyan and Zisserman, 2015]. Recently, Carter et al. [2019] used feature inversion to understand the InceptionV1 model (also called as "GoogLeNet") [Szegedy et al., 2015]. It is a 22 layer deep CNN that won the classification and detection task in the ILSVRC 2014 [Russakovsky et al., 2015]. Table 2.2 summarises all the global analysis methods discussed in this section.

Methods to analyse ML models locally

Another way to bring interpretability to black-box ML models is by generating explanations for their predictions [Montavon et al., 2018]. These explanations aim to highlight input features that contributed in favour of (or against) a decision. For instance, explanations for an image classification CNN highlight image pixels that influence model predictions. In literature, researchers sometimes refer to these explanations as *local explanations* as they explain individual model decisions [Ribeiro et al., 2016b]. These explanations assist in inspecting ML models by enabling humans to use domain knowledge to verify if a model is giving the right answers for the right reasons [Sturm, 2014]. This section discusses some methods to explain ML models locally.

Sensitivity analysis

One of the leading ways to explain decisions of ML models is by using *sensitivity analysis* (SA) that aims to analyse the sensitivity of model predictions on input features. SA takes different forms depending on the application, however, generally, it involves perturbing an input $\mathbf{x} \in \mathbb{R}^n$ by replacing it by $\mathbf{x} + \mathbf{g}$, where $\mathbf{g} \in \mathbb{R}^n$, and analysing its effect on the model prediction [Strumbelj and Kononenko, 2010, Krause et al., 2016]. For instance, for an image classification DNN that assigns a score $S_C(\mathbf{x}) \in \mathbb{R}$ to an input image $\mathbf{x}$ indicating its confidence that $\mathbf{x}$ belongs to the category C, SA assigns a relevance score to each input pixel suggesting that pixels with high relevance scores influence the prediction maximally.

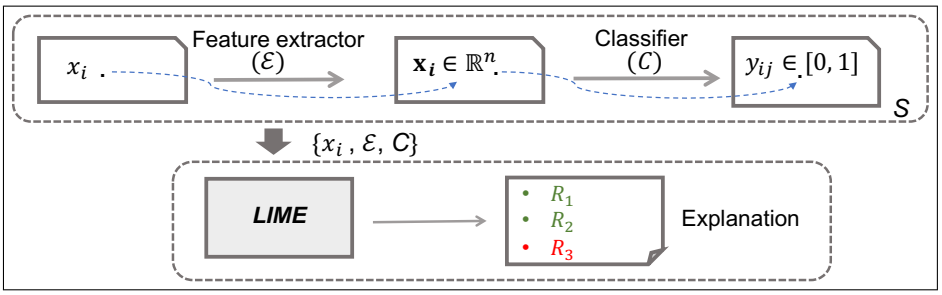

Figure 2.1: Schematic representation of LIME explaining why a machine learning system S applies label j to an instance x_i with probability y_{ij}.

Occlusion: One of the ways to perturb an input for SA is by changing input features in a pre-defined manner and analysing the change in model prediction. For instance, Zeiler and Fergus [2014] proposed to explain predictions of a DNN by sequentially occluding a group of input pixels using a sliding window and recording the change in model prediction on feeding the occluded image to the DNN. The group of input pixels that on occlusion results in the maximum change in model prediction is then the explanation of the prediction. Zintgraf et al. [2017] improved this methodology by combining prediction-difference analysis with conditional sampling and multivariate analysis. However, their approach is computationally intensive, and the explanations it generates are sensitive to the number of features it occludes [Ancona et al., 2018].

Sensitivity analysis with model approximation: Ribeiro et al. [2016b] proposed *LIME (local interpretable model-agnostic explanations)*, a method that tackles the problem of the number of features to occlude by segmenting inputs in human interpretable components (e.g., words in a text, super-pixels[9] in an image). LIME uses interpretable components to define *interpretable representations* of input data that assist in maintaining interpretability in the generated explanations. The generated explanations are easier to interpret because they show a more direct mapping between an input and its prediction. For example, for an e-mail classification model, LIME generates a list of words in an e-mail as an explanation for its classification to some category. Similarly, for an image classification model, LIME explains a prediction by identifying influential super-pixels in the input image.

Fig. 2.1 depicts an overview of what LIME aims to perform. LIME helps elucidate reasons for a system S applying label j to an input instance x_i with probability y_{ij}. LIME explains the prediction by listing three features (e.g., super-pixels): R_1, R_2, and R_3, where R_1 and R_2 influence the prediction positively (increase the prediction probability), and R_3 influences the prediction

[9]A super-pixel is a group of pixels sharing common characteristics (e.g., same intensity).

negatively (decreases the prediction probability).

LIME explains a prediction by approximating a complex model locally by an interpretable model using binary features that indicate the presence or absence of interpretable components. The interpretable model provides explanations in terms of its components. For instance, if the interpretable model is a linear model then weights and polarity of its features explain a prediction. Moreover, LIME is model-agnostic and hence useful to explain any ML model.

Formally, let $C : \mathbb{R}^n \rightarrow [0, 1]$ be a classifier, mapping a feature vector $\mathbf{x}_i = \varepsilon(x_i)$ to class label j, with probability $y_{ij} = C(\mathbf{x}_i)$. Define a sequence $\mathcal{X}_i$, which is composed of elements that are in some sense meaningful to the classification of x_i. For example, for a text classification model, $\mathcal{X}_i$ could be the sequence of unique words in an input and for an image classification model, $\mathcal{X}_i$ could be the sequence of super-pixels in an image. LIME defines an *interpretable space* $\mathcal{T} = \{0, 1\}^{|\mathcal{X}_i|}$ whose kth dimension corresponds to the kth element of $\mathcal{X}_i$. Then $\mathbf{x}'_i \in \mathcal{T}$ is the *interpretable representation* of x_i. Thus, LIME transforms an input instance x_i to a binary vector $\mathbf{x}'_i$ whose elements correspond to the presence and absence of elements of $\mathcal{X}_i$.

LIME defines an *(interpretable) explanation* as a model $g \in G$, where G denotes the class of interpretable models (e.g., linear models, decision trees). LIME learns the model g using a set of perturbed samples $\mathbf{z}'_{ik}$ and classifier predictions $C(\mathbf{z}_{ik})$, where $\mathbf{z}_{ik}$ represents the perturbed sample in the input space. LIME samples in the interpretable space by randomly setting dimensions of $\mathbf{x}'_i$ to the zero value. Formally, LIME learns g by the optimisation:

$$\arg\min_{g \in G} \left(L(C, g, \rho_{x_i}) + \Delta(g) \right) \tag{2.8}$$

where $L(C, g, \rho_{x_i})$ is a loss function that for an instance x_i measures how well the model g approximates the classifier C in the neighbourhood defined by ρ_{x_i}, and $\Delta(g)$ is a measure of model complexity (e.g., sparsity in the linear models). Thus, LIME minimises the sum of two functions to explain why C maps x_i to class label j. Algorithm 1 presents a version of the LIME algorithm that learns linear models to explain predictions of input instances. Chapter 4 introduces a methodology that extends LIME to machine listening models.

Gradient-based sensitivity analysis: Another way to perform SA is by computing the local gradient of the model function [Baehrens et al., 2010]. This approach avoids the expensive requirement of input perturbation and provides an automatic way to understand the sensitivity of model predictions on input features. Moreover, this local explanation approach is particularly useful for understanding DNNs as the backpropagation algorithm [Rumelhart et al., 1986, Goodfellow et al., 2016] can efficiently compute model gradients. Due to this,

Algorithm 1: Local explanation generation using LIME

Input: Classifier C, instance x_i and its interpretable representation $\mathbf{x}_i^{'}$
Input: Number of perturbed samples N_s
Input: Weight function ρ_{x_i}, number of interpretable components in the explanation N_{comp}
Output: Linear model $g(\mathbf{z}_i^{'})$ with weights $\mathbf{w}$

$Z \leftarrow \{\}$;
for $k \in \{1, 2, 3, ..., N_s\}$ **do**
  $\mathbf{z}_{ik}^{'} \leftarrow sample(\mathbf{x}_i^{'})$; `// sample randomly around` $\mathbf{x}_i^{'}$
  $Z = Z \cup \langle C(\mathbf{z}_{ik}), \mathbf{z}_{ik}^{'}, \rho_{x_i}(\mathbf{z}_{ik}) \rangle$;
end for
$\mathbf{w} \leftarrow learn\_linear\_model(Z, N_{comp})$;
return $\mathbf{w}$;

researchers have proposed several methods that perform gradient-based SA to generate local explanations for DNNs. This section discusses some of them below.

Simonyan et al. [2014] proposed to compute the magnitude of local gradients to explain predictions of a DNN. Their method creates a *saliency map*[10] by assigning a relevance score to each input pixel, suggesting that pixels with a high relevance score maximally influence a prediction. Formally, if for an input $\mathbf{x}$, $S_C(\mathbf{x})$ denotes the unnormalised class score for the class C, then the relevance score of the i_{th} pixel $R_i^C(\mathbf{x})$ for its influence on $S_C(\mathbf{x})$ is

$$R_i^C(\mathbf{x}) = \left| \frac{\partial S_C(\mathbf{x})}{\partial x_i} \right| \tag{2.9}$$

Although saliency maps provide some information about the local behaviour of a DNN model, they are very noisy (spatially discontinuous and scattered [Montavon et al., 2018]) and provide no information about the positive and negative evidence for a prediction due to the use of the absolute values of gradients [Ancona et al., 2018]. Thus, researchers have proposed several methods to improve the visualisations provided by saliency maps.

One category of methods focus on controlling gradients during the backward pass by only backpropagating the positive gradients [Zeiler and Fergus, 2014, Springenberg et al., 2015]. Zeiler and Fergus [2014] proposed their method by a different name (*Deconvolutional networks* (deconvnet)). However, Simonyan et al. [2014] later demonstrated that the Deconv method is a variant of gradient-based SA. Springenberg et al. [2015] named their method *guided backpropagation*. These methods generate high quality visualisations, but are only useful

[10] In literature, researchers refer to such maps by different names (e.g., attribution maps, heat maps).

for DNNs with ReLU non-linearity that scales an input x using the function $f(x) = max(0, x)$ [Krizhevsky et al., 2012]. Similarly, Selvaraju et al. [2017] proposed a method (they named it *Grad-CAM*) that first computes the gradient of a class score with respect to the deepest convolutional layer and then post-processes the gradients (by aggregation and applying ReLU non-linearity) to generate high-quality class activation maps. However, this method too is limited to the analysis of CNNs.

Another category of methods focuses on improving saliency maps without putting any additional restrictions about model architecture. For instance, Shrikumar et al. [2016] proposed to improve saliency map visualisation by preserving the signs of gradients from Eq. 2.9 and generating the relevance scores by multiplying the gradients with inputs. Following Ancona et al. [2018], this thesis refers to their method as *gradient* $\times$ *input*. The method generates $R_i^C(\mathbf{x})$ by

$$R_i^C(\mathbf{x}) = x_i \cdot \frac{\partial S_C(\mathbf{x})}{\partial x_i} \tag{2.10}$$

Sundararajan et al. [2017] extended the above method and proposed *integrated gradients* by averaging the signed gradients computed both at the input $\mathbf{x}$, and at other locations in the input space. The authors obtained these locations by varying $\mathbf{x}$ along a linear path starting from a user-defined baseline (e.g., an average of all inputs in a training dataset). Additionally, the authors also proposed two properties which they argue the attribution methods must satisfy.

Smilkov et al. [2017] demonstrated that noise in saliency maps is due to local variations in model gradients. They proposed a method (they call it *SmoothGrad*) that smoothes noise by first generating multiple inputs by perturbing an input locally through noise addition (e.g., Gaussian noise) and then averaging the saliency map from each input, resulting in visually improved saliency maps. Formally, if $R^C(\mathbf{x}) \in \mathbb{R}^n$ represents the saliency map for the classification of an input $\mathbf{x}$ to the category C, then SmoothGrad generates the noise-smoothed saliency map $R_S^C(\mathbf{x})$ by

$$R_S^C(\mathbf{x}) = \frac{1}{k} \cdot \sum_{1}^{k} R^C(\mathbf{x} + \mathcal{N}(0, \sigma^2)) \tag{2.11}$$

where $\mathcal{N}$ is the Gaussian distribution with standard deviation σ and k represents the number of locally perturbed samples.

Although SA helps in gaining some insights about the local behaviour of an ML model, there are some challenges. For instance, SA does not explain the score function $S_C(\mathbf{x})$, but provides knowledge about its (local) variations for

changes in the input features. For instance, as Montavon et al. [2018] noted, explanations using SA highlight features that make input more or less target class type, but do not highlight features that make an input belong to the target class. Thus, researchers have proposed several other methods to explain ML models locally. The sections below discuss some of them.

Function decomposition

This category of methods focuses on explaining the score function $S_C(\mathbf{x})$ by decomposing it in terms of the sum of the relevance scores for each input feature. One way to do that is by the *Taylor decomposition* of the score function [Montavon et al., 2018] that provides a tractable solution for special cases (e.g., deep ReLU networks). The score function for such cases is

$$S_C(\mathbf{x}) = \sum_{i=1}^{n} R_i^C(\mathbf{x}) \tag{2.12}$$

where $R_i^C(\mathbf{x})$ is the product of the input feature and the local gradient evaluated at that feature (see Eq. 2.10).

Another way to decompose the score function makes use of the network topology (e.g., the graph structure of DNNs). For instance, *layer-wise relevance propagation (LRP)* propagates the relevance of each neuron iteratively in the backward direction (from the output neuron to the input neuron in a DNN) using a set of pre-defined rules that follow the relevance conservation property [Bach et al., 2015].

Miscellanous

This category includes methods that do not come under the previous two categories. For example, the first method *counterfactual explanations*, aims to find the smallest change in the input features of an instance so that the model prediction for that instance changes to a pre-defined output (e.g., another class label) [Wachter et al., 2017, Sokol and Flach, 2019]. Thus, given an instance $\mathbf{x}$, a prediction (e.g., classifier) function f, and a pre-defined output y, the counterfactual explanation $\mathbf{e}$ is obtained by the constrained optimisation

$$\arg\min_{e} \max_{\beta} \beta(f(\mathbf{e}) - y)^2 + \theta(\mathbf{x}, \mathbf{e}) \tag{2.13}$$

where θ and β refer to a distance function and a weight parameter, respectively. Thus, the above optimisation aims to generate a new instance that is close to the input instance and whose classifier prediction is close to the pre-defined output.

The counterfactual explanations are easy to interpret as they highlight the

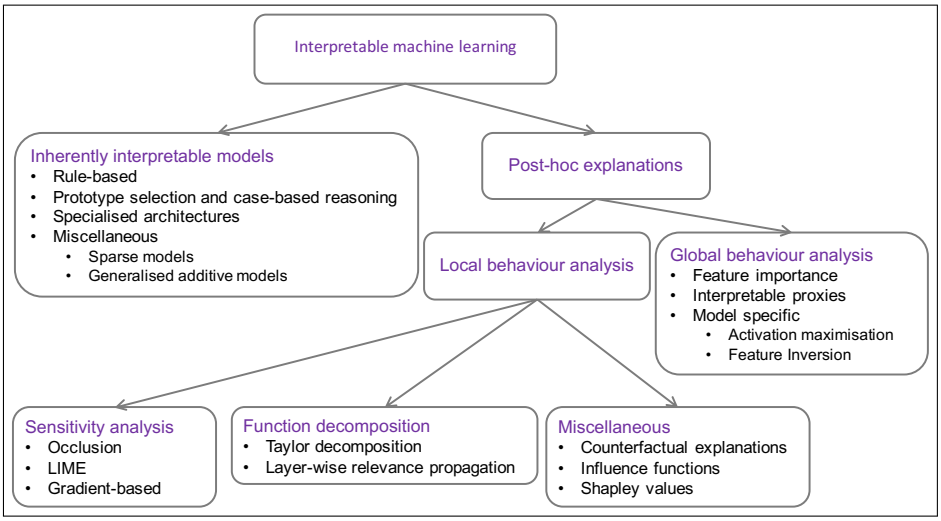

Figure 2.2: A high level taxonomy of some interpretable machine learning methods.

input features required to change a model prediction to a pre-defined output. However, these explanations have some limitations. For example, for an instance and pre-defined output, there exist multiple counterfactual explanations that may contradict with each other. In such a scenario, the process of selecting suitable counterfactual explanations for further analysis is not evident [Molnar, 2019].

Another way to analyse the local behaviour of ML models is by understanding how the training data instances influence model predictions. For example, to analyse the influence of an instance in a model prediction, one way is to remove or perturb the instance, re-train the model and analyse the change in prediction. However, doing this is expensive and thus Koh and Liang [2017] proposed to use *influence functions* that explain how the parameters of a model change when a training instance changes by a small amount. The authors demonstrated the versatility of influence functions for linear models and CNNs by using influence functions to identify dataset examples that maximally influence a prediction, to fix dataset errors, and to debug ML models.

Shapley values provide another way to explain the predictions of ML models [Lundberg and Lee, 2017, Merrick and Taly, 2019]. Shapley values come from game theory and provide a method to fairly distribute the total credit among players in a cooperative game. In the context of explaining model predictions, Shapley values provide a way to distribute the difference between a model prediction and a baseline (e.g., average prediction for all instances) fairly among the input features [Molnar, 2019]. Specifically, the Shapley value of a feature

i, denoted $\phi_i(v)$, can be interpreted as the weighted average of the marginal contribution of the feature to all possible coalitions of features.

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [v(S \cup \{i\}) - v(S))] \tag{2.14}$$

where N is a set of all K input features, $S \subseteq N$ represents a coalition of features, and v is a function that maps each S to a real value $v(S)$, where $v(\emptyset) = 0$.

Shapley-value-based feature attribution methods satisfy several desirable axioms. However, the computation of exact Shapley values is expensive and grows exponentially with the number of input features [Merrick and Taly, 2019]. There exist sampling-based approaches for approximating Shapley values, but they do not scale well to DNNs as they require multiple model evaluations [Ancona et al., 2019]. Thus, although Shapley values present a theoretically sound methodology for assigning attribution scores to input features, they have not been used for experiments in this thesis as the analysis of deep machine listening models forms a major part of this work.

Table 2.3 summarises all the local analysis methods discussed in this section. This section provided a detailed survey of different categories of IML methods that assist in analysing the behaviour of ML models. Fig. 2.2 summarises the discussed methods in a high-level taxonomy.

2.2.4 Interpretability in machine listening models

This section discusses some use cases where researchers analysed the machine listening models using some of the IML techniques from section 2.2.3. Compared to other domains (e.g., computer vision, text) relatively much less research exists about analysing the behaviour of machine listening models. However, recent years have seen an increased interest in applying IML methods to globally and locally understand machine listening models.

Global analysis

Sturm [2014] proposed to validate the performance of machine listening models by analysing their behaviour for inputs transformed through label preserving transformations which they refer to as "irrelevant transformations". Using the proposed transformations, they demonstrated that three state-of-the-art music information retrieval systems reported high performance by exploiting characteristics confounded with the ground truth.

Rodríguez-Algarra et al. [2016] performed intervention experiments to analyse a music genre classification model that uses scattering transform features to achieve the state-of-the-art performance on a publicly available genre classifica-

Categories	Methods	References
Sensitivity analysis	Occlusion	Zeiler and Fergus [2014], Zintgraf et al. [2017]
	LIME	Ribeiro et al. [2016b]
	Saliency maps	Simonyan et al. [2014]
	Deconvolutional networks	Zeiler and Fergus [2014]
	Guided backpropagation	Springenberg et al. [2015]
	Grad-CAM	Selvaraju et al. [2017]
	Gradient * input	Shrikumar et al. [2016]
	Integrated gradients	Sundararajan et al. [2017]
	SmoothGrad	Smilkov et al. [2017]
Function decomposition	Taylor decomposition	[Montavon et al., 2018]
	Layer-wise relevance propagation	[Bach et al., 2015]
Miscellaneous	Counterfactual explanations	Wachter et al. [2017], Sokol and Flach [2019]
	Influence functions	Koh and Liang [2017]
	Shapley values	Lundberg and Lee [2017], Ancona et al. [2019], Merrick and Taly [2019]

Table 2.3: Some methods to analyse the local behaviour of ML models.

tion dataset. The authors demonstrated that the model was non-trustworthy as it predicts by using information from inaudible frequencies (< 20 Hz).

Some researchers focussed on analysing the components (e.g., neurons, activation maps, filters) of DNN-based machine listening models. For instance, Dieleman and Schrauwen [2014] visualised the first convolutional layer filters in a deep music auto-tagging model trained using raw audio to discover that the model learns frequency-selective filters in that layer. Similarly, Schlüter and Böck [2014] analysed a scaled-down version of their deep music onset detection model by visualising the maximally activated feature maps and corresponding filters. They discovered some interesting insights into model functioning. They found that the model differentiates between the percussive and pitched onsets, and similar to spectral flux methods, it computes spectral differences over time.

The above analysis provides useful insights into the behaviour of both the machine listening models. However, the insights are restricted to shallow layer components as they are easier to interpret due to the proximity to input. Deeper layer components are much harder to interpret as they are multi-faceted [Nguyen et al., 2016b]. This thesis presents two methods (see Chapter 5 and Chapter 6) that help to analyse DNNs components without imposing any restriction on model depth.

Recently, researchers analysed the latent representations from deep machine listening models using activation maximisation with handcrafted priors. Specifically, Zhang and Duan [2018] performed AM in the input space using the L_2 norm of an input as the handcrafted prior to analyse a CNN that performs sound search by vocal imitation. Similarly, Krug and Stober [2018] performed AM in the input space using the L_1 and L_2 norms of an input as the handcrafted priors to analyse a CNN trained for predicting letters from input speech mel-spectrograms [Müller, 2015]. They demonstrated that performing AM in the input space using weak priors does not work well as it produces interpretable patterns only for lower layers of the CNN model. They proposed a novel global analysis method, *normalised averaging of aligned inputs (NAvAI)* that identifies features corresponding to a classification category by processing (aligning, averaging, and normalising) dataset instances of that category. The use of handcrafted priors limits the interpretability of synthesised examples and performing AM in the input space makes the input optimisation task difficult. Chapter 5 presents a flexible way to perform AM using a learned prior [Nguyen et al., 2016a].

Krug et al. [2018] proposed time-independent *neuron activation profiles* (NAPs) and demonstrated them to analyse the global behaviour of a fully-convolutional automatic speech recogniser trained to predict graphemes from input spectrograms. NAPs characterise network responses for groups of input

examples and are generated by averaging, normalising, and sorting neuron activations corresponding to a layer. The authors used NAPs to demonstrate that the speech recognition model predicts graphemes by using phoneme representations learnt in its hidden layers.

Local analysis

Some researchers have used saliency maps to identify the influential time-frequency bins in input audio representations (e.g., spectrograms). For instance, Choi et al. [2016] used saliency maps generated using the deconvolution method [Zeiler and Fergus, 2014] to explain predictions of a CNN-based music genre classification model. Moreover, in addition to visualising influential bins in input spectrograms, they auralised explanations by using the phase information of each input. Similarly, Schlüter [2016] used the saliency maps generated using the guided backpropagation method [Springenberg et al., 2015] to perform weakly-supervised classification of input mel spectrograms into vocal and non-vocal categories.

Saliency maps are helpful to gain some insight into the local behaviour of a model, but they are usually noisy [Smilkov et al., 2017]. Thus, to generate interpretable saliency maps, generally, some restriction is required (e.g., restricting the gradients, making use of special architectures [Zeiler and Fergus, 2014]), which limits their generalisation. Chapter 4 presents a model-agnostic local explanation method that aims to tackle these challenges.

This section discussed some machine listening models that researchers have analysed using IML methods and highlighted the some existing challenges in analysing those models. The next section introduces the machine listening use case this thesis uses for the experiments.

2.3 Singing voice detection

This section explains *singing voice detection* - the machine listening use case this thesis uses for experiments. Section 2.3.1 defines the SVD task and highlights why automatic SVD is challenging. Section 2.3.2 lists key SVD applications. Section 2.3.3 introduces common features used to train SVD models. Section 2.3.4 describes different approaches for SVD. Section 2.3.5 presents common metrics for evaluating the performance of SVD models. Finally, section 2.3.6 mentions current challenges and highlights future research directions for SVD.

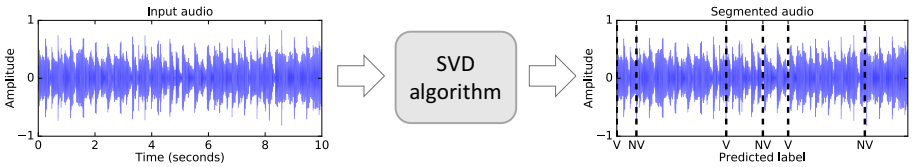

Figure 2.3: The figure depicts the application of a singing voice detection (SVD) algorithm for a 10-second musical audio clip (left) from "01 - A smile on your face.mp3" (time index : 32.0 seconds - 42.0 seconds) in the Jamendo dataset (see chapter 3). The SVD algorithm segments the input into temporal sections, indicating the presence or absence of singing voice (right). NV and V refer to the labels corresponding to the beginnings of non-vocal and vocal segments, respectively.

2.3.1 Definition

Singing voice detection refers to the automatic detection of the presence (or the absence) of the singing voice (or vocals) in short-duration (e.g., 200 ms) audio frames (or excerpts) [Humphrey et al., 2019, Lehner et al., 2018]. In literature, researchers sometimes refer to SVD by other names (e.g., vocal activity detection, vocal detection). The application of SVD to musical audio recordings labels the sequence of audio frames into vocal or non-vocal (instrumental music) sections. The vocal section indicates the presence of singing voice - either a capella or mixed with instrumental music. The non-vocal section indicates the absence of singing voice (or only the presence of instrumental music). Fig. 2.3 depicts the output from an SVD algorithm for a 10-second musical audio.

There exist many speech/music discrimination algorithms that detect the presence (or the absence) of speech in a musical audio frame. Schlüter and Sonnleitner [2012] and Papakostas and Giannakopoulos [2018] provide brief overviews of popular approaches. Although speech and singing voice have the same constituent, the human voice, popular speech/music discrimination algorithms perform poorly in detecting singing voice, due to fundamental differences between speech and singing [Berenzweig and Ellis, 2001, Ramona et al., 2008]. For instance, in comparison to speech, sung vocals have more timbral and pitch variations. Similarly, vocals have a comparatively higher correlation with background music than speech signals. Thus, there exists a need to design algorithms that focus on understanding singing voice and analysing how it differs from instrumental music. Current SVD algorithms make some assumptions about inputs and classification labels as mentioned below.

1. Singing voice in musical recordings may come from a single performer or several participants as in a choir or a mix of lead singer and backing vocals. An SVD algorithm labels sections containing any of these variants

Application	Overview	References
Singer identification	Identify the singer (artist) of musical audio	Tsai and Wang [2006], Berenzweig et al. [2002]
Singing voice extraction	Extract singing voice from polyphonic musical audio	Vembu and Baumann [2005]
Lyrics transcription	Identify lyrics from musical audio	Berenzweig and Ellis [2001]
Audio-to-lyrics alignment	Time-align musical audio with its lyrics	Humphrey et al. [2019], Wang et al. [2004]
Music discovery	Search for a particular "type" of song	Humphrey et al. [2019], Lee et al. [2018]

Table 2.4: Some music information retrieval applications that use singing voice detection as a preprocessing step.

of singing voice as vocals.

2. An SVD algorithm maps a temporal segment where a singer breathes between notes or where there is a short gap between notes to the non-vocal class.

3. An SVD algorithm focuses only on sung vocals and musical accompaniment. Hence, it assumes that an input musical audio does not contain any speech or rap segments.

4. An SVD algorithm makes no distinction between real and synthesised vocals.

2.3.2 Applications

Segmenting musical audio into vocal and non-vocal sections has widespread usage in music information retrieval (MIR) applications. In the majority of tasks, SVD acts as a preprocessing step, however in some, the SVD output is of direct use, and no further processing is necessary (e.g., commercial radio broadcasting [Ramona et al., 2008]). Table 2.4 mentions some MIR applications that use SVD as a preprocessing step.

2.3.3 Common features

This section briefly describes two features that are commonly used to train SVD models. Specifically, the section introduces the mel-spectrogram and mel-frequency cepstral coefficients (MFCCs) that are generally used as input features for training deep and shallow SVD models, respectively.

Mel-spectrogram

A mel-spectrogram is a time-frequency representation of an audio signal obtained by applying a mel-filterbank to summarise the energies across different frequency bands of a magnitude spectrogram (or a power spectrogram) [Müller, 2015]. The mel-filterbank is a set of triangular band-pass filters distributed along the mel-frequency scale, which is a perceptual scale of pitch describing the relationship between fundamental frequency and perceived pitch. The mel-scale is linear below 1000 Hz and logarithmic for higher frequencies. Thus, a mel-spectrogram provides more resolution for lower frequencies than higher frequencies. This thesis analyses the behaviour of a CNN-based SVD model that uses mel-spectrograms as input features (see Chapter 3).

Mel-frequency cepstral coefficients

MFCCs are short-term spectral features that model the spectral envelope of an audio signal, assisting in capturing the timbral properties of the audio [Müller, 2015, Jensen et al., 2009]. Researchers introduced MFCCs for speech signals [Davis and Mermelstein, 1980], however, they are also suitable for other types of audio signals (e.g., music [Jensen et al., 2009, Logan, 2000]). Some applications of MFCCs include speech recognition and speech synthesis [Jurafsky and Martin, 2009], musical instrument identification [Sturm et al., 2010], acoustic scene classification, and audio event detection [Stowell et al., 2015].

Extracting MFCC features from audio involves windowing (typically on the order of 10-100 msec), a mel-scale based smoothing of the log magnitude spectrogram, and a discrete cosine transform (DCT)-based decorrelation of the mel-spectral vectors. Although MFCC features are pseudo-invertible [Boucheron and De Leon, 2008], interpreting them in terms of the qualities of the underlying audio is difficult.

Fig. 2.4 depicts some visualisations from the MFCC extraction pipeline for a 2-second instance from the RWC dataset (see Section 3.2). The power spectrogram computation (Fig. 2.4 (b)) uses $l = 2048$, where l is the FFT size. The mel-spectrogram computation (Fig. 2.4 (c)) uses a filterbank of 128 filters from 0 Hz to $F_s/2$ Hz, where F_s represents the sampling frequency. On visually comparing the power spectrogram and the mel-spectrogram, it is evident that the mel-spectrogram is a smoother version of the power spectrogram and the mel-spectrogram emphasises the lower frequencies. The MFCC matrix (Fig. 2.4(d)) consists of 13 coefficients per audio frame computed using discrete cosine transform (DCT) Type-III [Oppenheim et al., 1999], where $k = 0$ represents the index of the zeroth MFCC. Appendix B explains the MFCC extraction and inversion procedure. This thesis analyses the behaviour of two tree-based SVD models

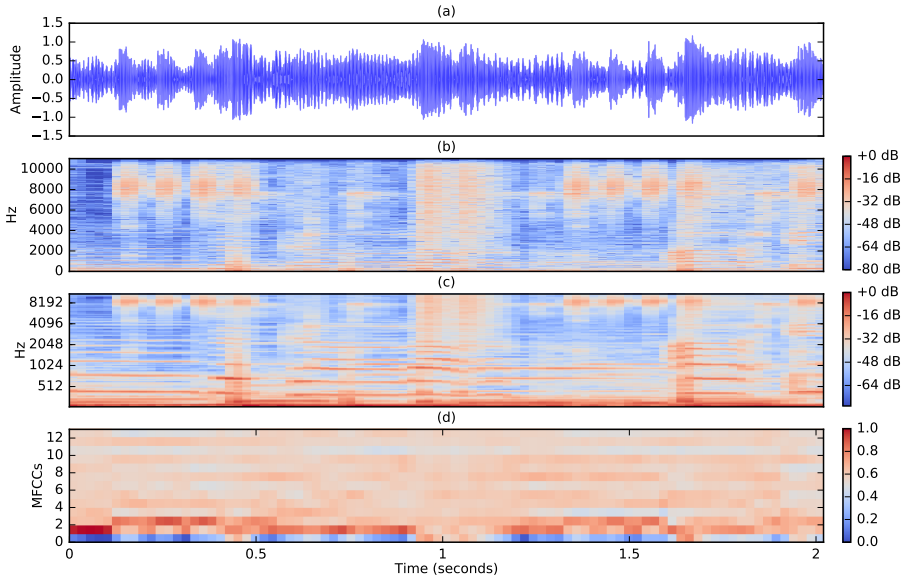

Figure 2.4: Visualisations of some representations from the MFCC extraction pipeline for a 2-second audio input from the RWC dataset. (a) Temporal representation, (b) Power spectrogram, (c) Mel-spectrogram, and (d) Normalised mel-frequency cepstral coefficients.

that use MFCCs as input features (see Chapter 3).

2.3.4 Methods

This section provides a brief survey of singing voice detection methods highlighting the present state-of-the-art. As this thesis does not focus on designing an improved SVD method, it selects methods with a viewpoint of highlighting how SVD research has evolved over the years.

Feature engineering-based methods

The majority of methods for designing SVD models fall under this category. These methods work by first extracting relevant acoustic features from input musical audio and then by training statistical machine learning classifiers using the extracted features. This thesis refers to SVD models designed with this methodology as 'shallow SVD models'.

The MIR research community has paid considerable attention to handcrafting acoustic features that help in discriminating vocals from non-vocals in musical audio and in selecting suitable machine learning classifiers to use the extracted features. This section focuses on highlighting leading features and machine learning algorithms for training shallow SVD models. For a more de-

tailed understanding of acoustic features and machine learning algorithms, see Müller [2015] and Bishop [2007], respectively.

In early efforts for training shallow SVD models, researchers used features from speech processing. For instance, the approach by Berenzweig and Ellis [2001] used three types of features: posterior probabilities of phonemes from a neural network trained for speech recognition, mel-frequency cepstral coefficients (MFCCs) [Davis and Mermelstein, 1980] and perceptual linear predictive coefficients (PLPCs). They used these features to train a hidden Markov model (HMM). Similarly, the SVD method by Nwe and Wang [2004] trained multi-model HMMs[11] using log frequency power coefficients (LFPCs) and the method by Vembu and Baumann [2005] combined MFCCs, PLPCs and LFPCs to train support vector machine (SVM) and neural network classifiers.

Later approaches focused on training shallow SVD models using hand-crafted features specific to musical audio. For instance, Nwe et al. [2004] noted that although both singing voice and instrumental music sounds are highly harmonic, the non-vocal sounds have more regular harmonic patterns compared to vocals. To utilise this property, they proposed sub-band based harmonic attenuated log frequency power coefficients (HA-LFPCs) that capture the energy distribution in sub-bands (different frequency bands) of the harmonically attenuated version of an input.

Nwe and Li [2007] demonstrated that perceptually motivated features (vibrato, harmonic content, singing formant, attack and decay envelope and timbre) are effective in discriminating vocals from non-vocals. Later, Regnier and Peeters [2009] also confirmed that analysing vibrato (frequency modulation) and tremolo (amplitude modulation) is useful in SVD. Their method does not train a model, but instead uses signal processing and simple thresholding (based on the extent value of vibrato and tremolo) to identify if a partial corresponds to a vocal sound. Rocamora and Herrera [2007] provide a detailed overview of then-popular hand-crafted features for SVD.

Ramona et al. [2008] argued that the performance of SVD methods highly depends on the quality of the extracted features. Their method builds a large training set by first extracting 116-dimensional features from preprocessed input musical audio frames over two time scales and then by using a feature selection algorithm to identify the 40 most discriminative dimensions in each feature vector. The authors used two time scales: a short time scale that extracts features from 32 ms frames with 16 ms overlap and a long time scale that extracts features from 1-second frames with 0.5-second overlap. They trained a one-vs-one SVM using the training set and post-processed the model predictions using

[11]The authors train multiple HMM models per classification category resulting in an HMM model space.

two approaches, a median filter and an HMM, where the latter performed better. Moreover, the authors also noted that the lack of publicly available datasets has restricted the comparison of different SVD algorithms and hence, they curated a dataset of 93 full-length songs from the Jamendo free music sharing website[12]. They made the dataset and ground truth annotations publicly available. We provide more details about this dataset in chapter 3, section 3.2.

Mauch et al. [2011] proposed three new acoustic features for SVD using the extracted predominant melody line: pitch fluctuation, MFCCs of the re-synthesised predominant voice, and the relative harmonic amplitudes of the predominant voice. They demonstrated that training shallow SVD models (an SVM-HMM) using a feature set of four acoustic features (MFCCs and the three features they proposed) helps to achieve high performance on a dataset of 102 pop music songs, 90 of which are from the publicly available RWC popular music dataset [Goto et al., 2002]. Moreover, the authors provided the ground truth annotations for all of the 100 songs in the RWC popular music dataset. Chapter 3, section 3.2, provides more details about the RWC dataset and reference annotations.

Lehner et al. [2013] trained a light-weight shallow SVD model only using MFCCs and their first-order derivatives to achieve high performance on both the publicly available (Jamendo and RWC) datasets. Specifically, they trained a random forest classifier [Breiman, 2001] with 128 trees using the first 30 MFCCs and their first-order derivatives extracted over an 800 ms observation window to classify a 200 ms audio frame. Moreover, the authors also compared the performance of their SVD model with three other models on the publicly available datasets demonstrating that appropriately selected MFCCs are sufficient in designing an SVD model with performance comparable to high performing complex models. The authors also noted that SVD models often incorrectly predict instruments with similar temporal and timbral characteristics as the singing voice (e.g., string instruments) to the vocal category.

To reduce the number of false positives for the vocal category (instrumental music sounds categorised as vocals), in subsequent work, Lehner et al. [2014] proposed three new acoustic features that they claim help in reducing the misidentification of sounds from pitch-continuous and pitch-varying musical instruments as the singing voice. Specifically, they proposed the *fluctogram* to characterise pitch fluctuations, *spectral contraction* to measure how much spectral energy lies in the centre and *vocal variance* to measure variance in the first five MFCCs (excluding the zeroth MFCC) over 11 audio frames. As the temporal context is important in SVD, the authors aggregated each of the three features over 40 audio frames to compute their variances. Their SVD model is a random

[12]http://www.jamendo.com

forest trained using 116-dimensional features. In addition to the new audio features, the authors use MFCCs and their first-order derivatives and the statistical means of the spectral flatness feature that estimates noise in an input. The authors evaluated the post-processed predictions from the SVD model demonstrating that the new audio features help to reduce the number of false positives by a considerable amount resulting in improved performance on the Jamendo and RWC datasets.

In follow-up work, Lehner et al. [2015] noted that including the temporal context in audio features by calculating variance over successive frames increases latency. Thus, to reduce latency-by-design, they proposed to train a unidirectional recurrent neural network (RNN) with long short-term memory (LSTM) units [Hochreiter and Schmidhuber, 1997, Goodfellow et al., 2016] that aggregates the temporal context in the input features. The authors trained the LSTM-RNN model using all but vocal variance features from Lehner et al. [2014], and the feature extraction happens over a smaller observation window (100 ms) with no temporal summary computation. The use of LSTM-RNN resulted in an SVD model that not only performed better than the one proposed by Lehner et al. [2014] on the Jamendo and RWC datasets but also reduced the maximum latency-by-design from 1500 ms to 140 ms.

The current state-of-the-art feature engineering method is the one proposed by Lehner et al. [2018]. Their method drastically reduces the number of features from Lehner et al. [2015] by just using 11 fluctogram features (post-processed using the two reliability indicators: spectral contraction, and spectral dispersion) and the first-order derivatives of the first 18 MFCCs. They use 29-dimensional feature vectors per frame to train an LSTM-RNN classifier with one hidden layer of 55 LSTM units. The authors trained, validated and tested their model's performance using a much larger and carefully curated dataset to make sure that its artefacts (e.g., album effect [Flexer and Schnitzer, 2010]) do not affect model performance. The dataset is available for research by contacting the authors. Table 2.5 summarises the feature engineering-based SVD methods.

End-to-end learning-based methods

Motivated by the success of deep learning in other pattern recognition tasks [LeCun et al., 2015], MIR researchers have experimented by training end-to-end deep learning models for SVD. Such models aim to automatically learn discriminative features for SVD from low-level input representations (e.g., mel spectrograms [Müller, 2015]). This thesis calls these models 'deep SVD models'.

Leglaive et al. [2015] trained a bidirectional LSTM (BLSTM) RNN model using low-level musical audio features. A BLSTM-RNN predicts by considering

Methods	Features	Classifiers
Berenzweig and Ellis [2001]	Posterior probabilities of phonemes, MFCCs, PLPCs	HMM
Nwe and Wang [2004]	LFPCs	Multi-model HMM
Vembu and Baumann [2005]	MFCCs, PLPCs, LFPCs	SVM, neural networks
Nwe et al. [2004]	HA-LFPCs	Multi-model HMM
Nwe and Li [2007]	Vibrato, singing formant, attack and delay envelope, timbre	-
Regnier and Peeters [2009]	Vibrato, tremolo	-
Ramona et al. [2008]	116 features (e.g., spectral centroid, zero crossing rate, MFCCs)	SVM
Mauch et al. [2011]	MFCCs, Pitch fluctuation, relative harmonic amplitudes of the predominant voice, MFCCs of the re-synthesised predominant voice	SVM-HMM
Lehner et al. [2013]	MFCCs, Δ-MFCCs	Random forest
Lehner et al. [2014]	Fluctogram, spectral contraction, spectral flatness, vocal variance, MFCCs, Δ-MFCCs	Random forest
Lehner et al. [2015]	Fluctogram, spectral contraction, spectral flatness, MFCCs, Δ-MFCCs	LSTM-RNN
Lehner et al. [2018]	Fluctogram post-processed with reliability indicators, Δ-MFCCs	LSTM-RNN

Table 2.5: Summary of the feature engineering-based singing voice detection methods. MFCCs: mel-frequency cepstral coefficients, PLPCs: perceptual linear predictive coefficients, LFPCs: log frequency power coefficients, HMM: hidden Markov model, SVM: support vector machine, HA-LFPCs: harmonic attenuated log frequency power coefficients, Δ-MFCCs: first-order derivatives of MFCCs, LSTM-RNN: unidirectional recurrent neural network with long short-term memory units.

both the present and past temporal context of input features [Goodfellow et al., 2016]. The input features are log-scaled mel spectrograms [Müller, 2015] that the authors extract from the vocal and percussive components of input audio. Their method generates these components by applying double-stage harmonic percussive source separation (HPSS) to input audio [Tachibana et al., 2010]. The input to the incrementally built three-layer deep BLSTM-RNN model is an 80-dimensional feature vector, in which the first and the last 40 features correspond to the vocal and percussive components, respectively. The deep SVD model even without post-processing the predictions reported better performance on the Jamendo test dataset than the state-of-the-art shallow SVD methods.

Schlüter and Grill [2015] used mel spectrograms to train a nine layer deep convolutional neural network (CNN). To tackle the problem of small training datasets available for SVD, the authors proposed seven data augmentation methods that they grouped into three categories. The first category consists of two data-independent methods: removing randomly selected bins and adding controlled noise in input excerpts. The second category includes four methods specific to audio data: pitch shifting, time stretching, loudness modification, and random frequency filtering of input excerpts. The third category includes a method specific to the binary classification task. The authors proposed to mix a vocal excerpt with randomly selected non-vocal excerpts, assuming this will help the CNN to learn invariance to the background music. The authors empirically found that pitch shifting performed best for their case and a CNN model trained with train and test time data augmentation is the state-of-the-art deep SVD model. The authors perform training data augmentation by using pitch shifting of $\pm 30\%$, time stretching of $\pm 30\%$, and random frequency filtering of ± 10 dB. Similarly, test data augmentation involves averaging model predictions over inputs and their $\pm 10\%$ and $\pm 20\%$ pitch-shifted versions.

Other methods

The majority of SVD methods require sub-second ground truth annotations to train shallow or deep SVD models. The process of creating ground truth is time-consuming, expensive, error-prone and in some cases, subjective. To avoid the need of sub-second annotations, some recent works in SVD use weakly-labelled [Schlüter, 2016] or unlabelled data [Pikrakis et al., 2016] to train vocal detection models that achieve performance on par with models trained using supervised learning.

Schlüter [2016] proposed a method to train deep SVD models using coarsely (weakly) annotated datasets. The author used a dataset of $10,000$ 30-second song snippets, each with a single ground truth label indicating the presence or

absence of vocals, to train three CNNs using multiple-instance learning [Foulds and Frank, 2010] and saliency maps [Simonyan et al., 2014]. Their method trains three CNNs sequentially by training the next CNN using the predictions or the summarised saliency maps of the previous CNN. The author evaluated all the CNNs by generating sub-second annotations for three test datasets, two of which (Jamendo and RWC) are publicly available. The results suggested that the SVD models trained using the proposed method performed comparably to an SVD model with the state-of-the-art architecture [Schlüter and Grill, 2015] trained using sub-second annotations.

Pikrakis et al. [2016] proposed to perform SVD using unsupervised learning that trains machine learning models using unlabelled data [Bishop, 2007]. Their method uses a sequence of short-term features extracted from input musical audio to learn a reasonably small dictionary using the K-singular value decomposition algorithm [Aharon et al., 2006]. The learnt dictionary through its constituents (atoms) is used to calculate a global threshold for the binarisation of the feature space. The authors evaluated their method on two publicly available datasets with different timbral characteristics using two types of features: the MFCCs and the bark band representation of a spectrogram [Rabiner and Schafer, 2010]. The two datasets are "Cante-100" which is a collection of 100 flamenco recordings, and a YouTube playlist of 19 Greek folk music songs. See Pikrakis et al. [2016] for more details about the datasets, ground truth annotations, and hosting repositories. The results suggested that a model trained using the bark band features with a 2-sparse representation and a dictionary of 64 atoms performed on par with a model trained using supervised learning on both the datasets. The authors did not compare the performance of their models against state-of-the-art method due to the use of different datasets.

2.3.5 Evaluation metrics

Lehner et al. [2013, 2014, 2015] evaluated their SVD models by calculating four metrics, namely, the % accuracy, precision, recall, and F-score for the vocal class. They defined the metrics as

$$\%\text{Accuracy} = 100 \cdot \left(\frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \right) \tag{2.15}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.16}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.17}$$

$$\text{F-score} = 2 \cdot \left(\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \right) \tag{2.18}$$

where TP is the number of true positives, FP is the number of false positives, TN is the number of true negatives, and FN is the number of false negatives for the vocal class.

Schlüter and Grill [2015], on the other hand, reported only three metrics for their SVD model. They calculated the classification error, recall and specificity for the vocal class. They defined error and specificity as

$$\%\text{Error} = 100 - \%\text{Accuracy} \tag{2.19}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{2.20}$$

Chapter 3 reports the performance of five SVD models used in this thesis using all six metrics from above. It is important to note that in addition to the above metrics one could also evaluate the models using threshold-free metrics (e.g., area under receiver operating curve, Brier score). However, this thesis does not include these details as this work in not aimed at performing an exhaustive evaluation of SVD models, instead, the work focusses on analysing the behaviour of high performing SVD models.

2.3.6 Research challenges

Although SVD is a well-explored MIR task and researchers have proposed multiple methods for it, there are still numerous issues with the existing approaches [Lee et al., 2018]. For instance, the majority of research in SVD uses musical audio of only a few genres (e.g., popular music) and there has been very limited work in exploring how well an SVD model generalises to musical genres not present in the training data [Scholz et al., 2017].

Similarly, recent research has shown that the state-of-the-art shallow and deep SVD models are susceptible to the loudness level of input musical audio due to the use of loudness sensitive features (e.g., zeroth MFCC) during model training [Lehner et al., 2018]. For instance, the authors noted that the number of false positives for the vocal category increases with increasing the loudness level of input audio. They demonstrated that using features that are insensitive to variations in audio loudness level (e.g., Fluctogram features) helps to develop loudness-invariant SVD models. Recently, Schlüter and Lehner [2018] demonstrated that using zero-mean convolutions in the first convolutional layer of a deep CNN forces the model to learn loudness-insensitive features.

2.4 Summary

This chapter provided a detailed survey of two research topics that this thesis deals with: interpretable machine learning, and singing voice detection. IML involves designing methods to analyse the behaviour of machine learning models. This chapter provided a detailed survey of the IML research field by highlighting the key definitions, motivating the need for interpretability in ML models, and describing popular IML methods. IML is a rapidly advancing research field, hence, describing all existing IML methods is challenging. Due to this, the methods this chapter described aimed at providing an insight into IML research directions. Moreover, for some methods, this chapter provides an in-depth survey as they form the basis of experiments this thesis describes in subsequent chapters.

This chapter also provided a detailed survey of the singing voice detection task - a machine listening application that deals with detecting the presence of vocals within musical excerpts. Specifically, the chapter defined the task, mentioned its key applications, provided a survey of key methods to train SVD models, and discussed current research challenges in designing SVD models.

This thesis uses a local explanation method (LIME) and two global analysis methods (activation maximisation and feature inversion) to analyse the behaviour of three SVD models. Chapter 3 provides more details about the SVD models and chapters 4, 5 and 6 describe the experiments that apply IML methods to analyse of the behaviour of the SVD models.

Chapter 3

Singing voice detection models

This thesis uses SVD as the machine listening use case for experiments (see Section 1.2). Chapter 2 described the use case and discussed key methods to develop an SVD model. This chapter discusses in detail the SVD models used in this thesis. In particular, this chapter discusses five SVD models: two shallow models and three deep models.

The organisation of the remainder of this chapter is as follows: Section 3.1 discusses the motivation behind the selection of SVD as an exemplar of machine listening, Section 3.2 introduces the datasets for the training, evaluation, and analysis of the SVD models. Section 3.3 describes the two shallow SVD models. Section 3.4 describes the three deep SVD models.

3.1 Motivation

Three main factors guide the selection of SVD as an exemplar for machine listening - the complexity of the use case, the performance and reproducibility of the state-of-the-art models and the interpretability of the classification categories.

Use case complexity: SVD is a fairly complex binary classification use case as sung vocals are highly correlated with background music and can exhibit several timbral and pitch variations. Moreover, certain musical instruments (e.g., electric guitars, violin) can mimic the temporal and timbral characteristics of vocals and negatively influence the precision of the vocal class (high false positives) [Lehner et al., 2013, 2014]. Finally, SVD datasets can have different musical textures (e.g., polyphony), further increasing the task complexity.

State-of-the-art models: SVD has been a topic of active research for

Dataset	N_{songs}	Duration (hours)	Ground truth
Jamendo	93	6.2	subsecond vocal and non-vocal annotations
RWC	100	6.8	subsecond vocal and non-vocal annotations
MedleyDB	122	7.2	vocal and non-vocal stems
ccMixter	50	3.2	vocal and non-vocal stems

Table 3.1: Overview of the singing voice datasets used for the design and analysis of the SVD models. N_{songs} refers to the number of musical audio files in a dataset. The duration column mentions the times calculated by Schlüter [2017, Table 9.1].

nearly two decades, leading to the development of different categories of methods and benchmarked datasets. In recent years, SVD models have become considerably accurate, with the high-performing models achieving accuracy close to 90% on benchmarked datasets. Moreover, multiple recent works have focussed on model reproducibility by either open-sourcing their implementation [Schlüter and Grill, 2015] or by explaining their methodology in detail [Lehner et al., 2013]. Thus, using SVD as a use case provides an opportunity of analysing high performing models (deep and shallow) that are reproducible with little effort.

Interpretable classes: To validate the behaviour of ML models, we need to verify whether the understanding from model explanations matches with our background knowledge. Interpretable classes assist in validating model behaviour as they help develop an understanding of how a model should be working. SVD has two highly interpretable classification categories - vocals and non-vocals and there exists a fair understanding about what distinguishes vocals from non-vocals. For example, the timbral properties of the singing voice are different from those of musical instruments.

Thus, we can consider SVD to be sufficiently rich and complex so as to serve as an exemplar for machine listening. Moreover, the interpretability methods proposed and demonstrated in subsequent chapters are fairly generic and are either directly applicable or easily extendible to other machine listening applications (e.g., multiclass classification). For example, Chettri et al. [2018] used SLIME (see Chapter 4) to explain the predictions of a spoofing detection model.

3.2 Datasets

This section describes the four singing voice datasets used for the design and analysis of the five SVD models. Table 3.1 provides an overview of the datasets.

- The **Jamendo** dataset consists of 93 full-length songs with Creative Commons license that Ramona et al. [2008] collected from the Jamendo free

music sharing website[1]. The musical audio files are stereo, sampled at 44.1 kHz and encoded using the Ogg Vorbis or mp3 standards. Each song is from a different artist, and the dataset contains songs of multiple genres.

Ramona et al. [2008] partitioned the dataset into subsets of 61 (training), 16 (validation), and 16 (testing) songs, respectively. The dataset is 6.2 hours in duration, of which the training dataset is 4 hours long [Schlüter, 2017]. Each song has subsecond annotations (by the same person) indicating the start and end of the vocal and non-vocal segments. The dataset and annotations were publicly available[2], but the weblink does not seem to be active now. This thesis uses the Jamendo dataset both for the design and the analysis of the five SVD models.

- The **RWC** popular music dataset, introduced by Goto et al. [2002], is one of the six subsets of the real world computing music database (RWC-MDB)[3]. The RWC popular music dataset contains 100 full-length popular music songs out of which 80 songs are in Japanese, and 20 songs are in English. The dataset is 6.8 hours in duration [Schlüter, 2017]. A single artist or a vocal group performed songs in the dataset. In total 15 male singers, 13 female singers, and 6 vocal groups performed 50, 44, and 6 songs, respectively, resulting in multiple songs per artist.

  Mauch et al. [2011] provided manually generated ground truth annotations for the RWC popular music dataset[4]. This dataset does not come partitioned into separate subsets for training and evaluating SVD models. Thus, researchers using this dataset have trained and evaluated their SVD models by performing 5-fold cross-validation [Mauch et al., 2011, Lehner et al., 2013, 2014, 2015, Schlüter and Grill, 2015]. This thesis uses a randomly selected subset of 20 songs from the RWC dataset for analysing the deep SVD models (see Chapter 4 and Chapter 6).

- **MedleyDB** is a multitrack dataset compiled by Bittner et al. [2014]. It consists of 122 professional or near-professional quality songs collected from multiple sources and freely available for use under Creative Commons license. The dataset songs are from 9 different genres, and the number of songs per genre is variable. The musical audio files are available in WAV format, sampled at 44.1 kHz with 16 bits per sample. The dataset and ground truth annotations are publicly available[5].

[1] https://www.jamendo.com
[2] http://www.enst.fr/~ramona/icassp08/
[3] https://staff.aist.go.jp/m.goto/RWC-MDB/
[4] https://staff.aist.go.jp/m.goto/RWC-MDB/AIST-Annotation/
[5] https://medleydb.weebly.com

The dataset provides three types of musical audio recordings for each song: unprocessed audio tracks, processed stems[6], and a multichannel mix. This thesis only uses a set of 20 randomly selected songs where each song contains both vocal and non-vocal stems and has no bleed between tracks. The thesis uses vocal and non-vocal stems and their mixes for analysing a deep SVD model (see Chapter 6).

- **ccMixter** collected by Liutkus et al. [2014] is a dataset of 50 full-length stereo songs processed from the ccMixter database[7]. The dataset is 3.2 hours in duration, contains songs of multiple genres, and is publicly available[8]. Each song in the dataset has three musical audio files in WAV format: a vocal track, a non-vocal track, and a mix of the vocal and non-vocal tracks. Similar to MedleyDB, this thesis uses this dataset for analysing a deep SVD model (see Chapter 4).

3.3 Shallow singing voice detectors

This section describes the two shallow SVD models that Chapter 4 uses for experiments. The method to design the models adapts the technique proposed by Lehner et al. [2013]. The thesis uses their technique as it only uses the mel-frequency cepstral coefficient (MFCC) features [Davis and Mermelstein, 1980] to train shallow SVD models with performance on public datasets comparable to more complex methods. This section describes the method to develop the shallow models in detail. The section first introduces features that the method extracts from musical audio files and then describes the two shallow SVD models and their training methodology using the extracted features. Finally, the section discusses the performance of both models on the Jamendo test dataset.

3.3.1 Input features

The method extracts a 60-dimensional vector from each frame of an input musical audio file. To do that, it first preprocesses (downsamples and downmixes) input to 22050 Hz mono audio. Then, it generates a time-frequency representation from the preprocessed audio by applying a fast Fourier transform (FFT) to 200 ms audio frames generated using 100 ms hop size and a 200 ms rectangular (boxcar) window. The selection of rectangular window is driven by the use of the shallow SVD models in an experiment that analyses their behaviour by generating temporal explanations for their predictions (see Section 4.3.1 for

[6] A stem combines multiple audio tracks of the same kind. For instance, a drum stem is a mix of kick drum, snare drum and high hat audio tracks.

[7] http://www.ccmixter.org

[8] https://members.loria.fr/ALiutkus/kam/

more details). Finally, the method uses a set of 30 mel-filters ranging from 0 Hz to 11025 Hz to extract 30 MFCCs (including the 0th) and their first-order derivatives from each audio frame.

The shallow SVD models perform classification over a 1-second excerpt. Hence, the method uses the approach proposed by Rocamora and Herrera [2007] to calculate the median and standard deviation of 60-dimensional vectors over ten frames and concatenate the results to generate a 120-dimensional feature. The method standardises the 120-dimensional feature by using the mean and standard deviation of features computed over the training dataset. The normalised 120-dimensional feature is an input to the SVD models. The method uses the LibROSA python package [McFee et al., 2015] for feature extraction.

3.3.2 Shallow models

The method trains two shallow SVD models. The first one is a binary decision tree (BDT) M_1. The method chooses a decision tree model as it is an interpretable model that can explain its predictions. The second SVD model is a random forest (RF) model M_2. The method chooses a random forest model as Lehner et al. [2013] used an RF model built with 128 trees to achieve high performance on publicly available datasets.

Training methodology

The method trains the shallow SVD models using features extracted from the Jamendo training dataset. The method selects model parameters by performing grid search in the parameter space. The method trains multiple models, one for each set of parameters and selects one that does not overfit to the training dataset and achieves the highest accuracy on the Jamendo validation dataset.

Specifically, the selection of the best BDT and RF models involves performing grid search to select three parameters: the maximum depth of the tree (d_{max}), the number of input features to use while searching for the best split (n_{features}), and the minimum number of samples required for a node to be a leaf node (n_{samples}). The set of values of the three parameters used for grid search is the same for both the models and is mentioned below.

- $d_{\text{max}} \in \{2, 4, 8, 16, 32\}$
- $n_{\text{features}} \in \{5, \log_2(d_{\text{features}}), 8, \sqrt{d_{\text{features}}}, 11, 14\}$, where d_{features} represents the dimensionality of input features
- $n_{\text{samples}} \in \{5, 6, 7, 8, 9, 10\}$

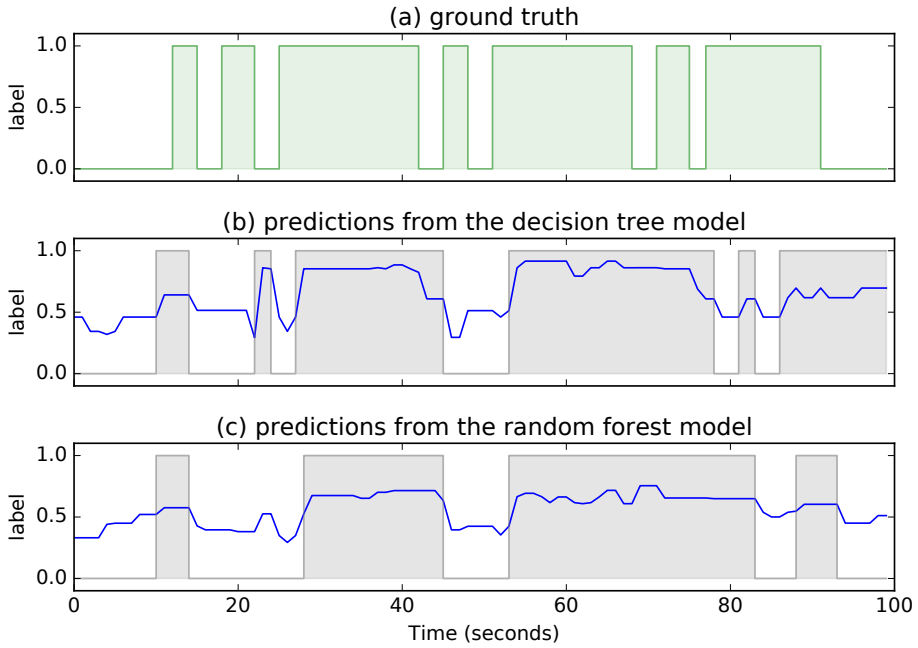

Figure 3.1: Figure depicting ground truth labels (0 : non-vocal, 1 : vocal) and predictions from the two shallow SVD models for a 100-second audio segment (time index : 60 seconds - 160 seconds) from the "04 - Inside.mp3" song in the Jamendo test dataset. The green coloured step plot depicts ground truth labels, the grey coloured step plots refer to prediction labels, and the blue coloured line plot depicts the vocal presence probability that each model assigns to an input excerpt.

Moreover, to train the RF model, in addition to the previous parameters, the method performs grid search to select an appropriate number of trees in the RF model (n_{trees}). The set of values used for the grid search of n_{trees} is $\{2, 4, 8, 16, 32, 64, 128\}$. The training procedure selects a split that maximises information gain and uses entropy as a metric for evaluating the quality of a split. The best BDT model (M_1) is 8 levels deep (d_{max}) with $n_{\text{features}} = 11$ and $n_{\text{samples}} = 10$. The best RF model (M_2) is 16 levels deep, has 64 trees in the forest with n_{features} and n_{samples} the same as M_1. The method trains all the models using the open-sourced scikit-learn library [Pedregosa et al., 2011].

Post-processing

Each model outputs the probability of a normalised 120-dimensional input feature containing singing voice. The model predictions are post-processed to reduce noise in predictions by applying a median filter with a window length of 7 frames (700 ms). Later, the smoothed predictions are converted to class labels

Models	Metrics	Jamendo dataset		
		Training	Validation	Test
Baseline	Accuracy	61.1%	63.3%	**57.5%**
Binary decision tree (M_1)	Accuracy	78.7%	75.4%	**71.4%**
	Error	21.3%	24.6%	28.6%
	Precision	0.823	0.801	0.722
	Recall	0.830	0.794	0.807
	Specificity	0.719	0.630	0.577
	F-score	0.826	0.784	0.745
Random forest (M_2)	Accuracy	91.6%	77.3%	**76.3%**
	Error	8.4%	22.7%	23.7%
	Precision	0.907	0.785	0.752
	Recall	0.960	0.865	0.876
	Specificity	0.846	0.590	0.595
	F-score	0.933	0.813	0.794

Table 3.2: Evaluation results of the three shallow singing voice detection models over the Jamendo test dataset. Baseline refers to a model that classifies all inputs to the vocal class. M_1 and M_2 refer to the best binary decision tree and random forest models, respectively.

by applying a threshold = 0.55. The values for the length of median filter and class threshold are taken from Lehner et al. [2013]. Figure 3.1 shows ground truth and predictions from the two shallow SVD models for a 100-second audio segment from the Jamendo test dataset.

3.3.3 Performance evaluation

The evaluation method evaluates M_1 and M_2 on the Jamendo test dataset using the six evaluation metrics from section 2.3.5. The method computes TP, FP, TN and FN for input excerpts of 1-second duration.

Table 3.2 reports the evaluation results of the best BDT and RF models. The table also reports the accuracy of a model (this thesis calls it 'baseline') that by default classifies all inputs to the vocal class. The accuracy of the baseline model reports the true class distribution within a dataset. For example, the vocal class occupies 61.1% and 63.3% of the Jamendo training and validation datasets, respectively. The table also reports the performance of M_1 and M_2 over the Jamendo training and validation datasets.

The best BDT and RF models achieve an overall accuracy of 71.4% and 76.3%, respectively on the Jamendo test dataset. The vocal class occupies 57.5% of the test dataset, suggesting that these two models have learnt some representation of singing voice that helps to detect vocals. It is important to note that researchers have reported designing more accurate shallow SVD models that achieve comparatively higher performance over the Jamendo test dataset [Ramona et al., 2008, Lehner et al., 2013, 2014]. However, this thesis

does not aim to design the state-of-the-art shallow SVD model, but to analyse whether the two reasonably accurate shallow SVD models are trustworthy.

3.4 Deep singing voice detectors

This thesis uses the SVD model (calls it 'SVDNet') introduced by Schlüter and Grill [2015]. The thesis chooses SVDNet as its source code is open-sourced[9] and it is state-of-the-art on publicly available datasets. SVDNet is a deep convolutional neural network (CNN) trained using time-frequency representations of musical audio excerpts. Schlüter and Grill [2015] proposed and applied different data augmentation techniques (e.g., pitch shifting) to improve the performance of the trained model. The following sections provide details about SVDNet's input features, architecture, training methodology and performance on the Jamendo test dataset.

3.4.1 Input features

The input to SVDNet is a normalised log-scaled mel-spectrogram excerpt of around 1.6-second duration extracted from a musical audio file. The method that Schlüter and Grill [2015] used generates an input in six steps.

1. In step one, the method pre-processes audio by downsampling to 22050 Hz and downmixing to mono.

2. In step two, the method computes a short-time Fourier transform (STFT) using frames of 1024 samples, a Hann window and a hop size of 315 samples and preserves the magnitudes of the resulting spectrum.

3. In step three, the method generates a mel-spectrogram by applying a mel-filterbank to summarise the energies across different frequency bands. The mel-filterbank is a set of band-pass filters distributed along the mel-frequency scale, which is a perceptual scale of pitch describing the logarithmic relationship between frequency and perceived pitch. The method uses 80 filters ranging from 27.5 Hz to 8000 Hz.

4. In step four, the method log-scales the mel-spectrogram by applying $x \rightarrow log(max(x, 10^{-7}))$ where x is the magnitude of a bin.

5. In step five, the method normalises each mel-band in the mel-spectrogram by scaling each bin value by the mean and variance of the corresponding mel-band calculated over the training set.

[9] Available at https://github.com/f0k/ismir2015

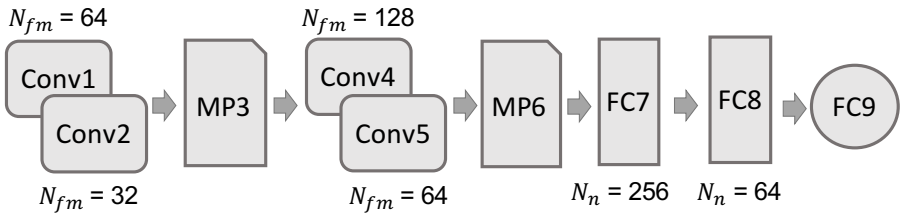

Figure 3.2: High-level architecture of the singing voice detection model introduced by Schlüter and Grill [2015]. N_{fm} denotes the number of feature maps in the output of a convolutional layer. N_n denotes the number of neurons in a fully-connected layer. Conv, MP and FC refer to the convolutional, max-pooling and fully-connected layers, respectively.

Layer	Input shape	Filter size	Stride	N_{fm}/N_n	Output shape	N_{params}
Conv1	$115 \times 80 \times 1$	3×3	1×1	64	$113 \times 78 \times 64$	640
Conv2	$113 \times 78 \times 64$	3×3	1×1	32	$111 \times 76 \times 32$	18464
MP3	$111 \times 76 \times 32$	3×3	3×3	-	$37 \times 25 \times 32$	-
Conv4	$37 \times 25 \times 32$	3×3	1×1	128	$35 \times 23 \times 128$	36992
Conv5	$35 \times 23 \times 128$	3×3	1×1	64	$33 \times 21 \times 64$	73792
MP6	$33 \times 21 \times 64$	3×3	3×3	-	$11 \times 7 \times 64$	-
FC7	$11 \times 7 \times 64$	-	-	256	256×1	1261824
FC8	256×1	-	-	64	64×1	16448
FC9	64×1	-	-	1	1	65

Table 3.3: SVDNet model architecture. Conv, MP and FC refer to the convolutional, max-pooling and fully-connected layers, respectively. Input and output shapes represent time $\times$ frequency $\times$ number of channels for the Conv layers. N_{fm}, N_n and N_{params} refer to the number of feature maps, number of neurons and number of parameters per layer, respectively.

6. Finally, in step six, the method generates excerpts from the normalised log-scaled mel-spectrograms by iteratively grouping 115 frames (around 1.6 seconds) using a hop size of 1 frame and (if necessary) zero padding.

3.4.2 Model architecture

SVDNet is a nine-layered CNN (see section 2.1.2) whose architecture is inspired from the VGGNet model [Simonyan and Zisserman, 2015]. VGGNet was the runner-up in the Imagenet Large-Scale Visual Recognition Challenge (ILSVRC) 2014 [Russakovsky et al., 2015]. It is one of the first models to successfully demonstrate that deeper CNNs have better prediction capability than shallow CNNs. It is a 16-layered deep CNN with a uniform architecture that has 3×3 convolutions and 2×2 max-pooling throughout the network.

Fig. 3.2 depicts key components of the SVDNet architecture and Table 3.3 provides a detailed description of the SVDNet architecture. The four convolutional layers (Conv1, Conv2, Conv4 and Conv5) perform convolutions using

64, 32, 128 and 64 filters, respectively. Each convolutional layer performs convolutions using 3×3 filters with 1×1 stride and no zero padding. The two max-pooling layers (MP3, MP6) perform 3×3 max-pooling (preserve maximum value) with 3×3 stride and no zero padding. The three fully-connected layers (FC7, FC8 and FC9) follow the last max-pooling layer. FC7 maps the input feature map to a 256-dimensional vector. FC8 is the deepest hidden layer in the network and FC9 is the output layer with a single neuron with sigmoid activation. The sigmoidal neuron outputs the probability of singing voice being present at the centre of an input audio excerpt by applying the sigmoid function $g(z) = 1/(1 + e^{-z})$ to the sum of the weighted sum of FC8 output and bias. All other layers in SVDNet use the leaky-ReLU non-linearity that scales an input x using the function $f(x) = max(0.01x, x)$ [Mass et al., 2013]. The total number of learnable parameters (weights and biases) in SVDNet is $1,408,225$.

In addition to being a scaled-down version, the SVDNet architecture differs from VGGNet in several other notable ways. For example, in VGGNet, inputs and outputs at a convolutional layer have the same spatial size, but in SVDNet due to the absence of zero padding, the spatial size shrinks progressively. Moreover, the two models differ in the configuration of the max-pooling layer. In VGGNet, the 2×2 max-pooling operation is performed with 2×2 stride, while SVDNet performs max-pooling by a factor of 3 with 3×3 stride. Finally, the two models also differ in how the number of convolutional filters changes after max-pooling layers. In VGGNet, the number of convolutional filters doubles after each max-pooling layer, but in SVDNet, the number of convolutional filters quadruples after the first max-pooling layer.

3.4.3 Model training

Schlüter and Grill [2015] trained SVDNet using mel-spectrogram excerpts of 115 frames extracted from the Jamendo training dataset and ground-truth labels denoting the presence or absence of vocals in the central frame. SVDNet is trained to predict the probability of vocal presence in the central frame of an excerpt using 57 frames on either side as context.

Before training, the network weights were initialised using random orthogonal matrices [Saxe et al., 2014]. The training method trains the network for a pre-fixed 20 epochs using mini-batches of 32 randomly sampled excerpts by minimising the binary cross-entropy loss between the ground-truth labels and network predictions. The parameters are updated using stochastic gradient descent with Nesterov momentum of 0.95 [Sutskever et al., 2013, Bengio et al., 2013] and an initial learning rate of 0.01 that is scaled by 0.85 after every 2000 updates.

The training method employs two methods to prevent overfitting. The first method sets the ground-truth labels to 0.02 (non-vocal) and 0.98 (vocal). Schlüter and Grill [2015] empirically discovered this approach to be better than L2 weight decay [LeCun et al., 1998]. The second method applies 50% dropout [Hinton et al., 2012, Srivastava et al., 2014] to the inputs of all fully-connected layers. The training method selects hyper-parameters depending on model performance on a validation set using an optimal threshold.

SVDNet was trained using three data augmentation methods to tackle the problem of smaller training dataset size. Specifically, the training method uses pitch shifting (±30%), time stretching (±30%) and random filtering (10 dB) to train SVDNet.

3.4.4 Post-processing

Given an input excerpt, SVDNet outputs the probability that the central frame in the excerpt contains singing voice. To reduce noise in SVDNet predictions, Schlüter and Grill [2015] post-process them by applying an 800 ms long median filter. The prediction labels are generated using an optimal threshold that is calculated using the Jamendo validation dataset. The optimal threshold is a value, from a predefined set of threshold values, that gives the least classification error on the Jamendo validation dataset. Once the optimal threshold is decided, SVDNet generates test dataset predictions, post-processes them using a median filter and applies the threshold to generate prediction labels for each input excerpt from the Jamendo test dataset.

3.4.5 Performance evaluation

Schlüter and Grill [2015] compared the ground-truth labels with SVDNet prediction labels for the Jamendo test dataset to compute three performance metrics for the vocal class: classification error, recall, and specificity. However, as discussed in section 2.3.5, in addition to the three evaluation metrics used by Schlüter and Grill [2015], this section also reports the accuracy, precision, and F-score for the vocal class. The evaluation method computes the six evaluation metrics from section 2.3.5 using TP, FP, TN, FN computed for input mel-spectrogram excerpts of around 1.6-second duration.

The first column in Table 3.4 mentions the performance of SVDNet on the Jamendo test dataset as reported by Schlüter and Grill [2015]. In the paper, the authors did not mention what threshold value they used to generate model predictions. Thus, the evaluation method replicates the prediction pipeline from the paper that generates predictions for the Jamendo test dataset using

the optimal threshold calculated over the Jamendo validation dataset[10]. The second column of Table 3.4 (corresponding to the label SVDNet-R0) mentions the evaluation results. The results suggest that the threshold value 0.66 helps the model to attain performance very close to what the authors reported in the paper. From here onwards, the performance of SVDNet (and SVDNet-R0) refers to the metrics after replicating the prediction pipeline.

The open-source code of SVDNet was developed using the Theano [Al-Rfou et al., 2016] and Lasagne [Dieleman et al., 2015] libraries. However, some experiments (see Chapter 5) require porting SVDNet to the Tensorflow library [Abadi et al., 2016]. SVDNet-R1 refers to the best of five models obtained by training the Tensorflow version of SVDNet. Table 3.4 reports the performance of SVDNet-R1 on the Jamendo test dataset. Results demonstrate that the new model is very close in performance to the SVDNet model with performance metrics differing in the order of 10^{-1}.

An experiment in Chapter 5 requires training another variant of SVDNet (the thesis calls it 'SVDNet-R2') by slightly modifying the architecture of SVDNet. The experiment replaces the single sigmoidal neuron in the output layer of SVDNet by a fully-connected layer with two neurons, each corresponding to one of the classification categories. The experiment does this to analyse whether representations that maximally activate the output layer neurons in SVDNet become more interpretable when a training methodology forces the model to learn individual class concepts. Additionally, the experiment applies a softmax layer to convert the vector of scores to a probability distribution over the classification categories. Similar to SVDNet-R1, the experiment uses the Tensorflow library for creating the computational graph. The experiment trains five models; SVDNet-R2 refers to the best of the five models. Table 3.4 reports the performance of SVDNet-R2 on the Jamendo test dataset. Results suggest that although the two-neuron model performs worse than one-neuron models, the decrement in performance is minimal. For example, the classification error from SVDNet-R2 predictions is only 1.1% larger when compared to SVDNet-R1.

3.5 Conclusion

This chapter presented five SVD models: two shallow models (a binary decision tree and a random forest) and three deep models (a deep CNN and its variants). For each model, the chapter discussed in detail its architecture, input features, training methodology and its performance on the Jamendo test dataset. The shallow SVD models are reasonably accurate while SVDNet is state-of-the-art

[10]The evaluation method uses the pre-trained model received through personal communication with Jan Schlüter.

	SVDNet	SVDNet-R0	SVDNet-R1	SVDNet-R2
Threshold	-	0.66	0.50	0.50
Error	**8.0%**	**8.1%**	**8.4%**	**9.5%**
Precision	-	0.901	0.896	0.890
Recall	0.914	0.926	0.925	0.907
Specificity	0.925	0.912	0.908	0.903
F-score	-	0.913	0.910	0.898
Platform	Theano & Lasagne	Theano & Lasagne	Tensorflow	Tensorflow

Table 3.4: Performance of SVDNet and its variants on the Jamendo test dataset.

on the test dataset. The chapter also introduced four publicly available singing voice datasets. This thesis uses these datasets for the design, evaluation and analysis of the SVD models. The following chapters discuss experiments that use interpretable machine learning algorithms to analyse the behaviour of the five SVD models presented in this chapter.

Chapter 4

SoundLIME

Chapter 2 discussed the two main ways for post-hoc interpretability of ML models: analysis of the global behaviour of a model, and explanation of the predictions of a model (local analysis). This chapter focusses on the local analysis of machine listening models. Specifically, this chapter introduces *SoundLIME (SLIME)*, a method to generate local explanations for predictions of any machine listening model, shallow or deep. Moreover, this chapter demonstrates the effectiveness of SLIME by using it to analyse the local behaviour of three singing voice detection models: two shallow models, and a state-of-the-art deep model (see Chapter 3).

This chapter consolidates the work from two conference publications [Mishra et al., 2017, 2020]. The first publication introduced SLIME and used it to analyse the local behaviour of the three SVD models [Mishra et al., 2017]. The second publication reported further experiments to analyse the behaviour of SLIME for different settings of the input parameters, highlighting an inherent weakness with the local explanation methods that perform sensitivity analysis by occluding input segments [Mishra et al., 2020]. Moreover, the publication also introduced a method to generate reliable explanations from SLIME.

The remainder of this chapter is organised as follows: Section 4.1 discusses the need for *interpretable explanations* for analysing machine listening models and highlights the key contributions of this chapter. Section 4.2 introduces the SLIME algorithm and describes the three categories of explanations it provides. Section 4.3 describes the experiments about applying SLIME to explain predictions of the three SVD models. Section 4.4 analyses the behaviour of SLIME for changes in the input parameters and discusses a method to generate reliable explanations from SLIME. Section 4.5 summarises the key results of this chapter. Finally, section 4.6 mentions the repositories hosting the open-sourced code and synthetic dataset used in this chapter.

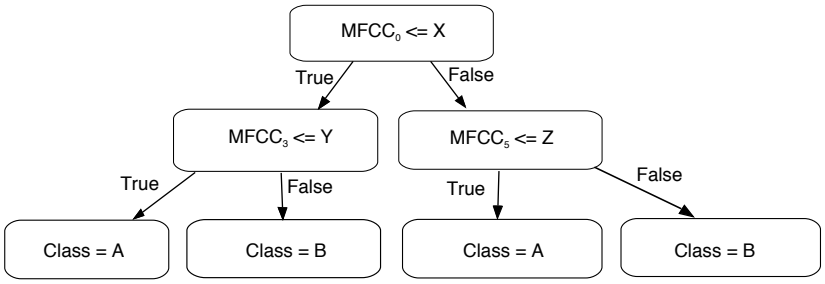

Figure 4.1: A binary decision tree for classifying audio using the values of three MFCC feature dimensions.

4.1 Introduction

Chapter 2 highlighted that one of the key ways to analyse ML models is by explaining their predictions. Explanations for model predictions are available either inherently (for interpretable models) or through the use of post-hoc methods (for black-box models). Local explanations, although useful, may not provide a clear insight into the behaviour of an ML model. For instance, in cases where an ML model is learnt using complex uninterpretable features, explaining model predictions in terms of those features may provide a limited understanding of the model behaviour. This issue is prevalent in applications using high-dimensional unstructured data (e.g., audio, text), as for them it is a common practice to transform input data into a set of low-dimensional features that are often uninterpretable. The example below discusses such a scenario for a machine listening system.

Consider a simple machine listening system , the classification component of which is the binary decision tree shown in Fig. 4.1. The input to this system is a T-sec audio excerpt, from which the system extracts D mel-frequency cepstral coefficients (MFCCs) [Davis and Mermelstein, 1980]. The system labels this D-dimensional feature vector as either "class A" or "class B" based on the values in specific feature dimensions. The system learns the particular dimensions (e.g., $MFCC_0$) and the thresholds of the decisions (e.g., X) through training.

As a binary decision tree is an interpretable classifier (see Section 2.2.3.1), we can explain predictions of the machine listening system in terms of the MFCC coefficients and thresholds. For instance, as shown in Fig. 4.1, if for an instance, the value of the zeroth MFCC is less than X and that of the third MFCC is less than Y, then the machine listening system classifies the instance to class A. Such an explanation helps explain the reasons for a prediction but provides limited insight into input audio qualities (e.g., energy, frequency content) influencing the prediction. This happens as the explanations are in terms of MFCCs that

provide limited transparency about the audio qualities they represent.

One might still roughly approximate the meaning of particular MFCCs: low MFCC dimensions (coefficients) relate to broad spectral structures (e.g., formants); high MFCC dimensions relate to fine spectral structures (e.g., pitch and harmonics); and the zeroth MFCC relates to the energy of a signal. However, as shown in Fig. 4.1, values along several MFCC dimensions and their thresholds jointly contribute to a prediction. This combination makes interpretation of local explanations even harder. For instance, it is ambiguous to understand what audible qualities the combination of the zeroth MFCC with either the third or the fifth MFCC captures. Thus, though the binary decision tree has clear decision rules, they are not easy to relate to audible qualities of inputs. With machine learning systems using black-box classifiers, e.g., deep neural networks (DNNs) or support vector machines (SVMs), the interpretation task becomes more challenging due to the lack of inherent interpretability. This motivates the use of interpretable explanations (see Section 4.2) for analysing the local behaviour of machine listening systems.

One of our key contributions is to propose and demonstrate a method for explaining predictions of any machine listening model. Specifically, this chapter describes SLIME, a method that extends a local explanation method (local interpretable model-agnostic explanations (LIME) [Ribeiro et al., 2016b]) to machine listening models. SLIME provides three ways to explain a prediction by pinpointing the time, frequency or time-frequency regions in an input that contribute most to a decision. This transforms a non-intuitive feature-based classifier decision into a more intuitive temporal and spectral description. This chapter discusses experiments that apply SLIME to three SVD models, demonstrating how the generated explanations are useful in gaining insight into model behaviour, and in identifying an untrustworthy model that fails to generalise.

Our other key contribution is to analyse the sensitivity of SLIME to changes in input parameters. This chapter describes experiments demonstrating that SLIME is sensitive to input parameter settings, an observation extending to LIME and other local explanation methods that perform sensitivity analysis by occlusion. Finally, this chapter discusses techniques to generate reliable explanations using SLIME. Some of these techniques apply to methods performing sensitivity analysis by input occlusion.

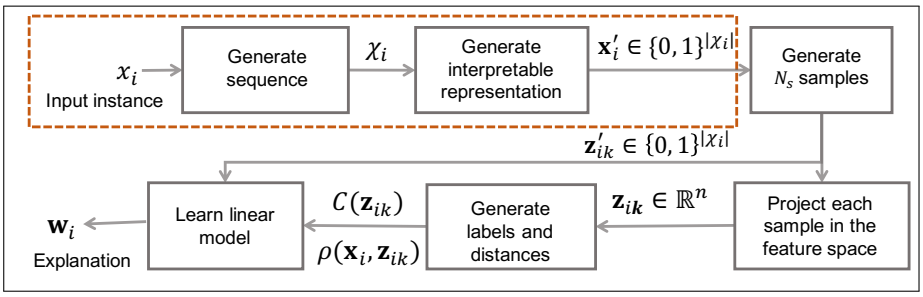

Figure 4.2: The functional block diagram of SLIME depicting the steps in generating explanation $\mathbf{w}_i$ for the prediction of an instance x_i.

4.2 Interpretable explanations for machine listening

This section describes the methodology to extend the local interpretable model-agnostic explanations (LIME) method proposed by Ribeiro et al. [2016b] to work with machine listening models.

4.2.1 Extending LIME to machine listening

Fig. 4.2 depicts the functional block diagram of SLIME. The algorithm consists of two components: the first one (the dotted box in Fig. 4.2) that defines interpretable sequences for audio is our contribution and the second one that uses these sequences to explain machine listening models comes from LIME. For details about the LIME algorithm, see Section 2.2.3.2.

SLIME explains the prediction of an audio instance x_i by defining three kinds of sequences: *temporal* $\mathcal{X}_i^t$, *spectral* $\mathcal{X}_i^f$ and *time-frequency* $\mathcal{X}_i^{tf}$. The algorithm terms each element of $\mathcal{X}_i^t$ a *super-sample* (terminology inspired by super-pixels in images), which the algorithm generates by the temporal partitioning of x_i. For example, SLIME uniformly segments the audio instance in Fig. 4.3 (a) into four super-samples, each notated T_i^n. Hence, $\mathcal{X}_i^t = (T_i^1, T_i^2, T_i^3, T_i^4)$. Similarly, SLIME generates each element of $\mathcal{X}_i^f$, notated A_i^m, by segmenting the magnitude spectrogram of x_i along the frequency axis. The magnitude spectrogram is the absolute value of the complex short-time Fourier transform matrix [Müller, 2015]. Hence, if SLIME segments the frequency axis into m spectral segments, $\mathcal{X}_i^f = (A_i^1, A_i^2, ..., A_i^m)$. Lastly, SLIME generates each element of $\mathcal{X}_i^{tf}$, notated B_i^p, by segmenting the magnitude spectrogram of x_i, both along the time and frequency axes. For example, in Fig. 4.3 (b), SLIME segments the magnitude spectrogram of the audio instance non-uniformly into eight time-frequency blocks. Hence, $\mathcal{X}_i^{tf} = (B_i^1, B_i^2, ..., B_i^7, B_i^8)$. SLIME terms the elements of the

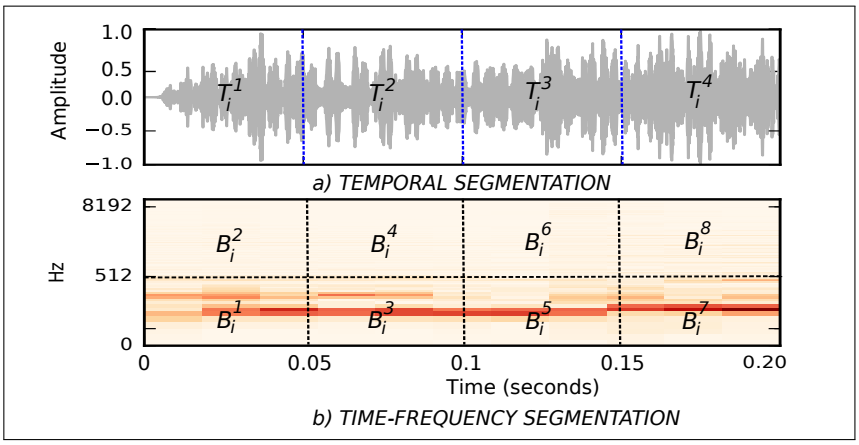

Figure 4.3: Input segmentation-based sequence generation for SLIME. (a) Temporal segmentation of the audio instance x_i into four super-samples (T_i^n), each of duration 50 ms. (b) Time-frequency segmentation of x_i into 8 time-frequency blocks (B_i^p). Similarly, segmenting the magnitude spectrogram in (b) only along the frequency axis will generate spectral segments.

interpretable sequences as *interpretable components*. Thus, for the temporal sequence, each interpretable component is a super-sample, and for the spectral and time-frequency sequences, each interpretable component is a spectral segment and time-frequency block, respectively.

The next step is to map an input instance x_i with a feature representation $\mathbf{x}_i \in \mathbb{R}^n$ to its interpretable representation $\mathbf{x}'_i \in \{0,1\}^{|\mathcal{X}_i|}$. SLIME uses each of the three interpretable sequences to define an interpretable space $\mathcal{T}$ (e.g., $\mathcal{T}^t$ is the temporal interpretable space) and an interpretable representation $\mathbf{x}'_i$. This creates three interpretable representations for an input instance x_i. SLIME denotes the temporal, spectral and time-frequency interpretable representations as $\mathbf{x}_i^{t'}$, $\mathbf{x}_i^{f'}$ and $\mathbf{x}_i^{tf'}$, respectively. These representations provide three ways of understanding a prediction, each highlighting the temporal, spectral or time-frequency segments in an instance maximally influencing its prediction.

To find the temporal explanation, SLIME locally approximates a classifier $C : \mathbb{R}^n \rightarrow [0,1]$ with a linear model $g_t(\mathbf{z}_i^{t'}) = \mathbf{w}_t^T \cdot \mathbf{z}_i^{t'}$, where $\mathbf{z}_i^{t'} \in \mathcal{T}^t$ represents the temporal interpretable representation of a synthetic sample. g can be any interpretable model, however, in this thesis g is a linear model. SLIME generates N_s synthetic samples from $\mathcal{T}^t$ in a way that depends on $\mathbf{x}_i^{t'}$, i.e., randomly occluding or setting to zero the dimensions of $\mathbf{x}_i^{t'}$. For example, for the audio instance in Fig. 4.3 (a), one possible $\mathbf{z}_i^{t'} = (1,0,1,0)$. This synthetic sample indicates the absence of super-samples T_i^2 and T_i^4. There exist 2^{N_c} unique synthetic samples for an interpretable sequence with N_c elements. SLIME projects each synthetic sample $\mathbf{z}_i^{t'}$ to the input space, computes the corresponding weighting

factor $\rho_{\mathbf{x}_i}(\mathbf{z}_{ik}^t)$, and generates the prediction $C(\mathbf{z}_i^t)$. The weighting factor measures how structurally similar a synthetic sample is to the input audio instance. SLIME learns the linear model g_t by minimising the sum of the squared loss and model complexity (see Eq. 2.8) over this dataset of synthetic samples and their probabilities. Formally, denote the temporal interpretable representation of the kth synthetic sample as $\mathbf{z}_{ik}^{t'}$ and the feature representation of its input space projection as $\mathbf{z}_{ik}^t$. Define a weight function $\rho_{\mathbf{x}_i}(\mathbf{z}_{ik}^t) : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$. In this thesis, the weight function is an exponential kernel (width ω) defined over a distance function Γ, i.e., $\rho_{\mathbf{x}_i}(\mathbf{z}_{ik}^t) = e^{\frac{-\Gamma(\mathbf{x}_i, \mathbf{z}_{ik}^t)^2}{\omega^2}}$ [Ribeiro et al., 2016b]. SLIME computes the locally-weighted loss as

$$L(C, g, \rho_{x_i}) = \sum_{(\mathbf{z}_{ik}^{t'}, \mathbf{z}_{ik}^t) \in Z^t} \rho_{\mathbf{x}_i}(\mathbf{z}_{ik}^t)[C(\mathbf{z}_{ik}^t) - g(\mathbf{z}_{ik}^{t'})]^2 \tag{4.1}$$

Similarly, SLIME randomly samples $\mathbf{x}_i^{f'}$ and $\mathbf{x}_i^{tf'}$ to learn the spectral and time-frequency linear models, respectively. Each of these models provides explanations in terms of their learned weights. The magnitudes of the model weights relate to the importance of interpretable components (e.g., super-samples, time-frequency blocks) in the classification of x_i. Thus, if w_1 and w_2 denote the weights corresponding to super-samples T_1 and T_2, respectively, then $|w_1| > |w_2|$ implies super-sample T_1 influences the prediction more than T_2. Similarly, the polarity of weights refers to the correlation between an interpretable component and the classifier prediction. For example, if $w_1 < 0$ and $w_2 > 0$, then the temporal segments T_1 and T_2 are respectively negatively and positively correlated with the classifier prediction. The weight function ρ_{x_i} controls the influence of each synthetic sample in the learned model. Thus, a distant sample in the interpretable space will have a lower contribution to g, facilitating robust learning. For pseudocode of SLIME, refer to Algorithm 1 corresponding to LIME. For example, to generate temporal explanations using SLIME, replace $\mathbf{x}_i^{'}$ with temporal interpretable representation $\mathbf{x}_i^{t'}$ in Algorithm 1.

4.3 Demonstration

This section describes experiments that demonstrate SLIME for analysing the local behaviour of machine listening models. Specifically, the experiments involve explaining the predictions of three singing voice detection (SVD) models - two shallow models and one deep model. Chapter 2 defines the singing voice detection task, and Chapter 3 describes the three SVD models in detail, discussing their architectures, training methodologies, and performances on the Jamendo test dataset. Each of these models classifies an audio excerpt into

two categories: non-vocal (instrumental music without singing voice) and vocal (instrumental music with singing voice). This section first describes the experiments to analyse the local behaviour of the shallow models. This section then explains the experiments about analysing the deep SVD model.

4.3.1 Explaining predictions of the shallow vocal detectors

The two shallow SVD models designed using the binary decision tree and random forest classifiers achieve accuracies of 71.4% and 76.3%, respectively, on the Jamendo test dataset (see Chapter 3, Section 3.3.3). The vocal category occupies 57.5% of the Jamendo test dataset, which suggests that the two shallow SVD models may have learnt some representation of singing voice that helps them to detect vocals. One way to validate this hypothesis is by analysing if the models make vocal predictions by utilising information from the vocal content in an input, i.e., to analyse if the models are trustworthy [Sturm, 2014].

The temporal explanations from SLIME assist in validating the above hypothesis by identifying super-samples (time segments) that maximally influence a prediction. Thus, to analyse the trustworthiness of the two models, the experiment involves generating temporal explanations for inputs classified to the vocal category by both the models and validating (for each input) whether the maximally influencing super-samples contain singing voice.

It is important to revisit that this experiment uses a rectangular window in the feature extraction pipeline of the SVD models. This is due to the reason that other window functions (e.g., Hann) will influence the energy of super-samples in an instance and this may result in the SVD models becoming less sensitive to the content of super-samples with near-zero energy. The use of rectangular window avoids this issue.

To generate temporal explanations, SLIME segments a one-second audio instance uniformly into ten super-samples. SLIME first generates $N_s = 1000$ synthetic samples in the temporal interpretable space $\mathcal{T}^t$. SLIME decides a suitable value for N_s depending on the stability of explanations (see Section 4.4.1). SLIME then approximates each model's decision boundary in the neighbourhood of the instance by a linear model that it learns in the temporal interpretable space. SLIME performs this approximation using an exponential weighting function with $\omega = 25$ and the cosine distance as the distance function Γ. The minimum number of interpretable components (super-samples) sufficient to explain a prediction may vary from one instance to the other. This happens as the selection of influential interpretable components depends on the linear model weight distribution. However, to reduce the model complexity ($\Delta(g)$ in Eq. 2.8), this experiment generates explanations with a pre-fixed number of

Index	Duration (seconds)	Vocal probability		Temporal explanations		Ground truth
		BDT	RF	BDT	RF	
41	1.0	0.97	0.85	$6, 7, 9$	$2, 0, 7$	$0 - 9$
178	1.0	0.86	0.86	$9, 8, 4$	$9, 6, 0$	$0 - 9$
58	0.4	0.80	0.76	$6, 5, 3$	$0, 2, 6$	$0 - 3$
124	0.4	0.92	0.84	$0, 4, 6$	$6, 9, 8$	$6 - 9$

Table 4.1: Temporal explanations from SLIME for audio instances from "03 - Say me Good Bye.mp3" in the Jamendo test dataset. Index: instance index; Duration: duration of the vocal segment; Vocal probability: model confidence about the presence of singing voice; Temporal explanations: top-3 super-samples maximally influencing a prediction; Ground truth: super-samples containing singing voice; BDT: binary decision tree; and RF: random forest.

super-samples. To do this, SLIME first uses the dataset of synthetic samples and corresponding model (BDT or RF) predictions to select the top-3 super-samples (by using the forward selection algorithm) and then learns a linear model using the selected super-samples [Ribeiro et al., 2016b]. Forward selection is a type of step-wise regression that assists in feature selection [Hastie et al., 2009].

Table 4.1 reports the temporal explanations generated by SLIME for four instances from the file "03 - Say me Good Bye.mp3" in the Jamendo test dataset. For each audio instance, the "temporal explanations" column lists the super-sample indices in decreasing order of their influence on model predictions. The magnitude of the linear model weights corresponding to each super-sample determines the influence of the super-sample in the prediction. The analysis of the temporal explanations helps to gain some insight into how the SVD models form their predictions. For example, both the models classify instance 41 to the correct category, but the temporal explanations for both the predictions are very different. Similarly, both the models correctly predict that instance 178 contains singing voice, but again for very different reasons. The auralisation of all the temporal explanations for instances 41 and 178 reveals an interesting insight into the BDT model behaviour. The majority of super-samples in the temporal explanations contain a strong instrumental onset sound in addition to the singing voice. This is intriguing as it may happen that instead of "listening" to the singing voice in those super-samples, the BDT model predicts by using information from the instrumental sounds.

To further investigate the above observation, the experiment applies SLIME to explain the predictions of audio instances (from the same test file) that contain instrumental music and singing voice in non-overlapping temporal sections. Instances 58 and 124 are two such instances that contain singing voice in the first and last 400 ms, respectively. The temporal explanations for the BDT model predictions for both the instances assist in revealing that although the

model is highly confident about the instances containing singing voice, the super-samples maximally influencing the predictions mostly contain instrumental music. Such an observation calls into question the generalisation capability of the BDT model. The temporal explanations for the RF model predictions for both the instances suggest that the model is behaving expectedly, as it predicts by using information from singing voice segments. This experiment demonstrates the usefulness of SLIME in identifying an untrustworthy model.

Temporal explanations assist in understanding the local behaviour of a machine listening model, however, there are some limitations. For example, to generate interpretable temporal explanations, SLIME either requires input audio to be of sufficient temporal duration or requires some restriction on the number of super-samples in an audio instance. Moreover, for instances containing singing voice and instrumental music for its complete duration (e.g., instance 41), temporal explanations highlight super-samples that maximally influence a prediction, however, they provide no information about the content in those super-samples that affects the prediction. One way to understand this is by analysing the spectral or time-frequency explanations from SLIME. These explanations highlight the influential spectral regions which may assist in gaining some more insight into the model behaviour. The next section discusses the time-frequency explanations in detail.

4.3.2 Explaining predictions of the deep vocal detector

This section demonstrates the time-frequency explanations from SLIME for analysing the local behaviour of the CNN-based SVD model proposed by Schlüter and Grill [2015] (see Chapter 3). Moreover, this section qualitatively compares the time-frequency explanations from SLIME with saliency maps that are local explanations for DNN predictions generated using gradient-based sensitivity analysis [Simonyan et al., 2014, Zeiler and Fergus, 2014, Springenberg et al., 2015] (for more details about saliency maps, see Chapter 2). This section skips the demonstration of spectral explanations as one can easily generate them by first segmenting an input time-frequency representation into spectral regions and then following the time-frequency explanation generation steps.

The deep SVD model assigns predictions to mel-spectrogram excerpts of around 1.6 seconds duration indicating the probability that the excerpts contain singing voice. SLIME explains the predictions of the deep SVD model by mapping input excerpts to their time-frequency interpretable representations (see Section 4.2). SLIME performs the mapping by segmenting the time-frequency axis of input excerpts into six temporal and four spectral segments. Such temporal segmentation keeps the temporal duration of the resulting time-frequency

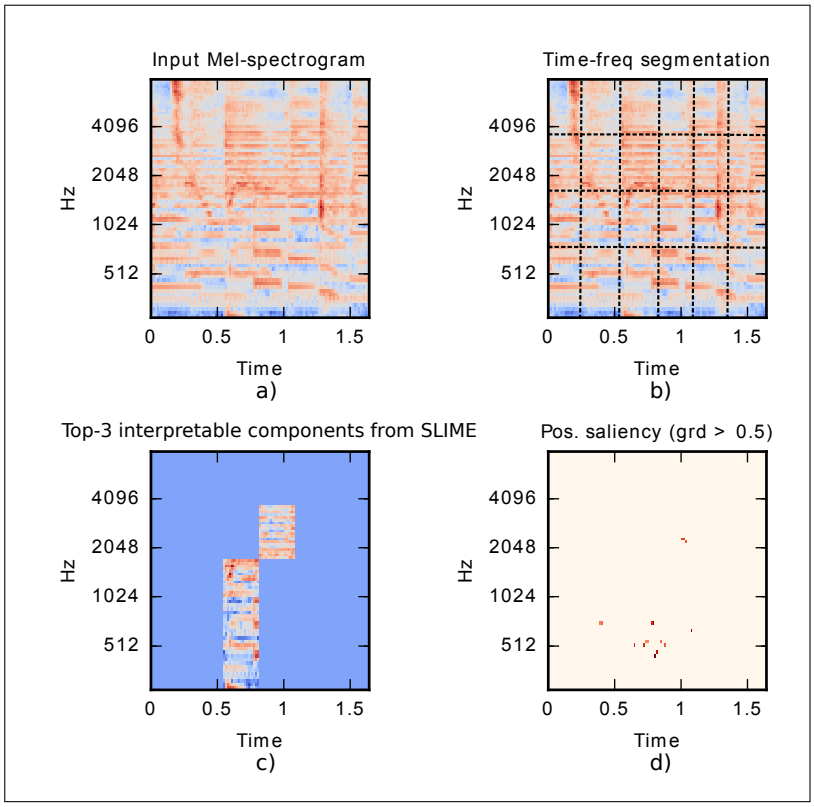

Figure 4.4: The time-frequency explanation generation from SLIME. (a) mel-spectrogram representation of a 1.6 second input audio excerpt from "03 - Say me Good Bye.mp3" in the Jamendo test dataset (time index: 122.5 seconds - 124.1 seconds, confidence score = 0.96), (b) time-frequency block generation through input segmentation, (c) the positive time-frequency explanation for the input highlighting the three most influential interpretable components, (d) the normalised thresholded positive saliency map for explaining the input prediction.

interpretable components sufficiently long to facilitate their auralisation (after inversion to the temporal representations, see Appendix B). The duration of the first five temporal segments is 266 ms each, and that of the last segment is 280 ms. The input segmentation along the frequency axis generates four spectral segments, each containing 20 spectral bins. Hence, SLIME maps each input mel-spectrogram excerpt x_i to a time-frequency sequence $\mathcal{X}_i^{tf} = (B_i^1, ..., B_i^{24})$, where the j_{th} time-frequency block B_i^j represents the j_{th} dimension in the time-frequency interpretable space $\mathcal{T}^{tf}$. Fig. 4.4 (a) and (b) depict the mel-spectrogram representation and its time-frequency segmentation, respectively, for an excerpt from "03 - Say me Good Bye.mp3" in the Jamendo test dataset. The deep SVD model correctly classifies the excerpt to the vocal class with probability = 0.96.

SLIME generates a time-frequency explanation by synthesising 2000 samples through random sampling from the interpretable space, approximating the non-linear decision boundary by a linear model using the L_2 norm as the distance measure, and selecting three interpretable components (time-frequency blocks) with the top-3 positive weights. This thesis calls such explanations positive time-frequency explanations as they highlight time-frequency blocks that positively influence model predictions. Fig. 4.4 (c) depicts the positive explanation for the prediction of the audio excerpt that contains a mixture of singing voice and instrumental music in its first 900 ms and only instrumental music in the last 700 ms. The auralisation of the positive explanation (after inversion to the temporal domain, see Appendix B) reveals that all the time-frequency blocks in the explanation contain singing voice. Such an observation raises trust in the model and its predictions. Moreover, the experiment reveals that all the time-frequency blocks in the negative explanation have temporal indices greater than 1 second. The negative explanation includes the time-frequency blocks with top-3 negative weights. This indicates that the time-frequency blocks containing only instrumental music negatively influence the classifier prediction. This also seems to be the correct behaviour. Thus, the time-frequency explanations help to understand what sections in the input are maximally influencing the prediction.

The experiment also qualitatively compares the time-frequency explanations from SLIME with saliency maps that, like the time-frequency explanations, are tools to analyse deep neural network models. Saliency maps highlight the influence of input dimensions on DNN predictions by using the backpropagation algorithm [Rumelhart et al., 1986, Goodfellow et al., 2016] to compute the local gradient of the model function [Simonyan et al., 2014]. For details about different methods to generate saliency maps, refer to Chapter 2, Section 2.2.3.2. This experiment uses the saliency map generation method from Zeiler and Fer-

gus [2014] as the authors demonstrated the method generates high-quality visualisations for CNNs. The authors introduced their method as a function decomposition method that uses a deconvolutional network (deconvnet) to map CNN activations to the input space. However, the method is the same as computing the local gradients of the model function, with an additional constraint that prevents the negative gradients from backpropagating [Simonyan et al., 2014, Springenberg et al., 2015]. The experiment compares the positive time-frequency explanations with the positive saliency maps that highlight the input dimensions that positively influence the model predictions. The local gradient of the model function at these dimensions is greater than zero. To highlight the most influential input dimensions, the experiment refines the positive saliency maps by selecting dimensions with normalised gradient greater than 0.5. The experiment generates such maps only for the output layer of the deep SVD model. Fig. 4.4 (d) shows the thresholded positive saliency map for the input excerpt in Fig. 4.4 (a).

It is important to note that saliency maps highlight individual dimensions in an input while the explanations from SLIME are the time-frequency blocks (a group of input dimensions). One way to compare the two model analysis approaches is by visually verifying whether the time-frequency blocks highlighted by SLIME capture all the input dimensions highlighted by saliency maps. A visual comparison of the example in Fig. 4.4 shows that the explanations from SLIME include most of the key dimensions highlighted by the saliency map. Another way to compare the two explanation methods is by measuring how many dimensions highlighted by a saliency map are enclosed in the explanation generated by SLIME. Formally, if in a normalised thresholded saliency map, the total number of non-zero input dimensions = d_{total} and the number of non-zero input dimensions that overlap with SLIME explanations = $d_{overlap}$, then the % agreement between the two explanation techniques is given by $\%Agreement = \left(\frac{d_{overlap}}{d_{total}}\right) \cdot 100$. For the audio excerpt in Fig. 4.4, this agreement is 62.5%.

An experiment expands the above quantitative comparison to a set of 1349 randomly chosen excerpts from the Jamendo test dataset. Specifically, for each song in the dataset, first, a random integer $N_{\text{excerpts}} \in [10, 25]$ is sampled. This integer indicates the number of excerpts to be sampled from that song. Then, N_{excerpts} are sampled randomly from different time indices in the song. The experiment applies this sampling procedure to sample five batches of excerpts with the total number of excerpts=1349. Further, for each batch of random excerpts, agreement% is computed between positive SLIME explanations and positive thresholded saliency maps. Table 4.2 mentions the number of excerpts

Batch index	$N_{\text{instances}}$	Average agreement
1	269	44.52%
2	254	46.07%
3	298	48.34%
4	261	45.32%
5	267	48.29%

Table 4.2: Average agreement% between positive SLIME explanations and thresholded positive saliency maps for randomly sampled batches with $N_{\text{instances}}$ audio excerpts.

and the average agreement for each random batch.

The results suggest that on average, the positive explanations from SLIME achieve 46.50%($\pm$1.55%) numerical agreement when compared with the thresholded (normalised gradient > 0.5) positive saliency maps. Moreover, the instance-based analysis of the comparison results reveals that for some instances, the numerical agreement is 100%, but there are instances where the agreement is less than 10%. An explanation for the instances with a poor agreement between the two methods is that for those instances the decision boundary in their neighbourhood is highly non-linear, and approximating such a boundary with a linear model will result in poor explanations from SLIME.

This experiment does not perform an exhaustive comparison (by varying the preset factors, e.g. threshold, the number of interpretable components) between the two techniques. The above analysis aims to provide an estimate of the performance of model-agnostic SLIME against a model-dependent technique for some preset values. It is obvious that the numerical agreement will be high if the constraints are softer and vice versa. Saliency maps assist in highlighting the input dimensions that are influential to model predictions, but the explanations can suffer from the lack of temporal context around the highlighted dimensions. On the other hand, one can readily invert the explanations from SLIME to an acoustic form for auralisation, which may provide additional insights into how a model is forming its prediction for an input.

4.3.3 Discussion on the number of synthetic samples (N_s)

LIME and SLIME generate local explanations using N_s synthetic samples in the interpretable space $\mathcal{T}$ (see Section 4.2.1). For LIME, Ribeiro et al. [2016b] mention the number of synthetic samples used in their experiments but do not discuss why those values are suitable for their experiments. A discussion about the number of synthetic samples N_s is important for two reasons. First, N_s affects the *time* T_s LIME (or SLIME) takes to generate an explanation. Second, N_s influences the *stability* of local explanations from LIME (or SLIME).

Ideally, an explanation should remain the same even on multiple iterations of applying LIME (or SLIME) to the same instance. The indices and polarity of interpretable components in an explanation must remain the same, however, their order or weights may change. However, empirical results demonstrate that interpretable components in LIME (or SLIME) explanations do change on multiple iterations. This happens because both the methods sample randomly in $\mathcal{T}$. Thus, this section seeks to understand the influence of N_s on the stability of local explanations from SLIME and on the time SLIME takes to generate each explanation.

The experiment uses the trained model, dataset, and the SLIME setup from section 4.3.2. The experiment involves first randomly selecting five excerpts from each audio file in the Jamendo test dataset (see Chapter 3) and then applying SLIME to explain the prediction of each excerpt in a batch of 80 by highlighting the top-k interpretable components per explanation (the experiment uses $k = 3$ and 5). Moreover, the experiment involves iterating this process five times, each time randomly sampling 80 audio excerpts, and generating explanations to highlight the top-k interpretable components in each explanation.

This experiment defines the stability of an explanation to be inversely proportional to the number of unique interpretable components U_n from the sequence $\mathcal{X}_i^{tf}$ that appear in explanations generated with m iterations. For example, if on applying SLIME $m = 2$ times to an audio excerpt to highlight the top-3 time-frequency segments in each iteration, the explanations are denoted as sets $\xi_1 = \{B_1, B_2, B_3\}$ and $\xi_2 = \{B_2, B_6, B_5\}$, then $U_n = 5$, as B_2 appears twice in the six components. To understand the effect of N_s on the stability of explanations from SLIME, this experiment generates five explanations for each of the 80 excerpts in the randomly sampled batches. For each excerpt in a sampled batch, U_n is computed over the aggregated set of five explanations. Fig. 4.5 (a) reports the results of the experiment. The plot depicts the average value of U_n over the five batches of excerpts for different values of N_s, where $N_s \in \{100, 250, 500, 1000, 2000, 5000, 10000\}$. The error bars represent standard deviation of the mean distribution corresponding to the sampled batches.

The result shows that for both the cases (top-3 and top-5), U_n decreases with increase in N_s, for smaller values of N_s, but nearly stabilises for larger values of N_s. The result also demonstrates that to generate stable explanations, SLIME does not need to perform an exhaustive search of the interpretable space $\mathcal{T}$ as although the maximum number of synthetic samples is 2^{24}, the explanations become stable for $N_s = 5000$ and $N_s = 10000$ samples for the top-5 and top-3 cases, respectively.

The above experiment also records the average time (T_s) SLIME takes in generating an explanation for a given N_s. Fig. 4.5 (b) reports the results for

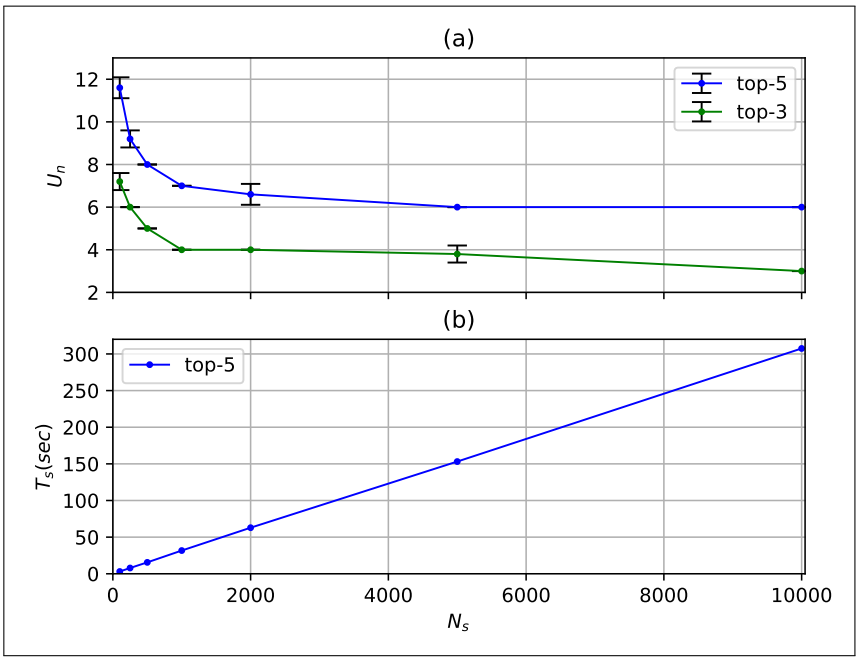

Figure 4.5: Plotting the influence of the number of samples (N_s) on (a) the stability of local explanations and, (b) the time SLIME takes in generating an explanation. U_n denotes the number of unique interpretable components, and T_s denotes the time SLIME takes (in seconds) in generating an explanation.

the experiment that applies SLIME to the batches of randomly selected audio excerpts to highlight the five most influential time-frequency segments in each excerpt. The results for the top-3 case are similar to the top-5 case, hence, for the better visualisation, the figure shows results only for the top-5 case.

The experiment runs SLIME on a computer with 1.6 GHz Intel Core i5 processor and 8 GB memory. The results show that T_s increases linearly with N_s, reaching around five minutes for an explanation generated with $N_s = 10000$. The reported time includes the time taken for prediction by the CNN model. Importantly, T_s largely depends on the prediction time of the CNN model and the sampling time of SLIME, hence, the plot does not report error bars as T_s is independent of sampled excerpts. These results suggest that selecting a suitable N_s depends on the trade-off between the stability of an explanation and the time taken to generate it. In the above experiments, $N_s = 1000$ seems to be a good trade-off.

4.4 Analysing the robustness of SLIME

This section describes experiments to analyse whether SLIME explanations are sensitive to the synthetic sample generation process. SLIME samples in the interpretable space by randomly occluding (masking) one or more interpretable components in an input. SLIME occludes interpretable components by replacing them with synthetic components. For example, in the previous experiments (see Section 4.3), SLIME occludes interpretable components by replacing them with the same size synthetic components with dimensions (e.g., bins in a time-frequency block, samples in the temporal representation) set to the value zero. This section analyses if the explanations SLIME generates are sensitive to how SLIME occludes the interpretable components, i.e., whether SLIME explanations change with changes to the masking content (the content of the synthetic components). This analysis will be useful not only to SLIME but also to other methods that use the input occlusion step in their explanation pipeline (see Zintgraf et al. [2017], Ribeiro et al. [2016b], and Zeiler and Fergus [2014]).

This section first describes the experiment to select an appropriate value for the number of synthetic samples N_s. This section then describes experiments to analyse SLIME's behaviour for five types of input perturbations, each modifying the masking content. Finally, this section introduces and demonstrates a method for deciding the content of synthetic components.

4.4.1 Selecting an appropriate N_s

The SLIME algorithm is sensitive to the number of synthetic samples it generates in the interpretable space. Section 4.3.3 demonstrated this for the SVDNet model (see Chapter 3, Section 3.4.5) using a dataset of 80 randomly selected audio excerpts. In that experiment, SLIME generates stable explanations ($U_n = 6$ and 4 for the top-5 and top-3 cases, respectively) for $N_s \geq 5000$. This experiment (analysing the robustness of SLIME) uses the SVDNet-R1 model (see Chapter 3, Section 3.4.5) and a comparatively much larger dataset of audio excerpts. Thus, before analysing SLIME's behaviour for different types of input perturbations, the experiment re-computes an appropriate N_s using the new model and the new random subset of audio excerpts. The new appropriate value of N_s will make sure that any changes in SLIME explanations due to different input perturbation methods are not the results of using an unsuitable N_s.

The experiment applies SLIME to explain the predictions of SVDNet-R1 for a dataset of 25 randomly selected audio excerpts from each audio file in the Jamendo test dataset. Thus, the dataset size in this experiment is 400 audio excerpts, five times larger than the experiment in Section 4.3.3. SLIME

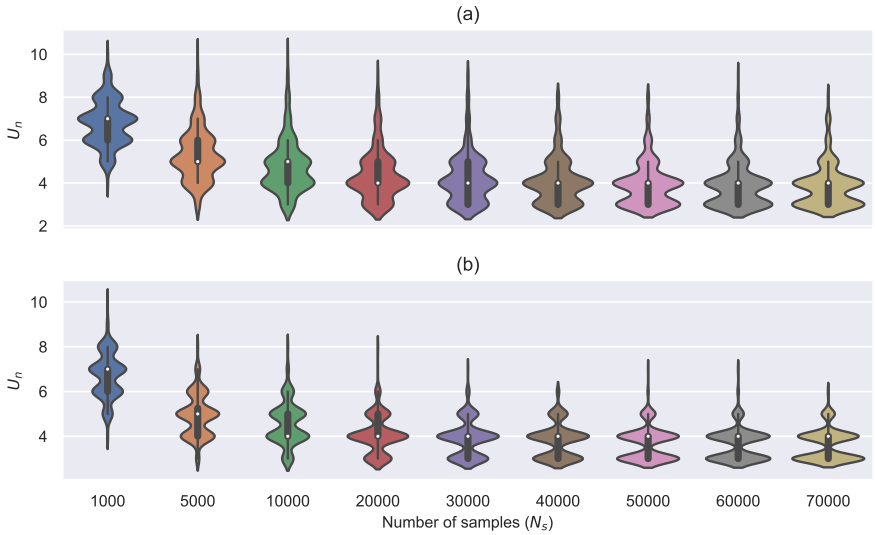

Figure 4.6: Plotting the influence of the number of samples (N_s) on the stability of local explanations for audio excerpts from the (a) Jamendo dataset and (b) RWC dataset. U_n denotes the number of unique interpretable components.

explains a prediction from SVDNet-R1 temporally by segmenting an input mel-spectrogram along the x-axis into 10 temporal segments and highlighting the three most influential interpretable components (top-3) positively or negatively influencing a prediction. This is another type of temporal explanation where instead of segmenting an input in the time domain, SLIME segments it temporally in the time-frequency domain. Thus, for the mel-spectrogram in Fig. 4.4, this refers to segmenting the time-frequency representation only along the x-axis (temporal axis). Moreover, SLIME occludes interpretable components by synthetic components with all bins set to the value zero, does not perform any normalisation to the synthetic samples, and uses the L_2 norm as the distance measure. Similarly to Section 4.3.3, for each N_s, the experiment computes U_n using an aggregated set of explanations that it generates by applying SLIME five times to the same audio excerpt. The experiment uses nine different values of N_s.

Fig. 4.6 (a) depicts the results of the experiment. The results show that similar to the experiments in Section 4.3.3, SLIME explanations become stable for higher N_s values. Moreover, for the new model and dataset, an appropriate value of N_s seems to be $\geq$ 50000, as for those values the median of the U_n distribution is four with a high likelihood of sampling $U_n = 3$.

The next step is to repeat the above analysis using audio excerpts from the RWC dataset to analyse if the conclusions from above generalise to other

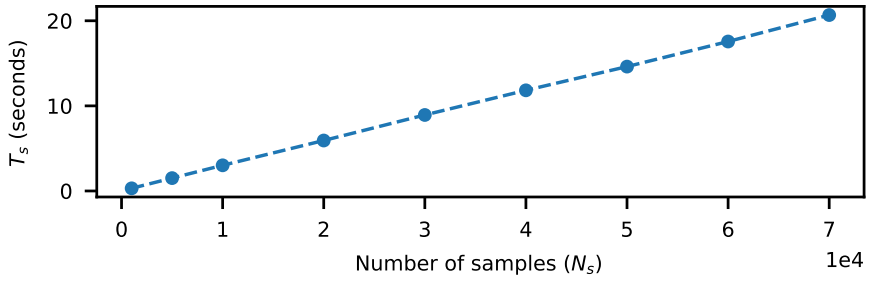

Figure 4.7: Plotting the influence of the number of samples (N_s) on the time T_s (seconds) SLIME takes in generating an explanation for audio excerpts from the Jamendo dataset.

datasets. A dataset of 500 audio excerpts is randomly selected from 20 audio files in the RWC dataset (see Chapter 3). Fig. 4.6 (b) shows the results of the experiment for the RWC dataset. Similar to Jamendo, an appropriate N_s value seems to be ≥ 50000. Additionally, for the experiment with the Jamendo dataset, Fig. 4.7 plots the average time SLIME takes to generate an explanation for different N_s values. The timing information comes from running the experiments on a Linux machine with Intel(R) Xeon(R) Gold 5122 processor @ 3.60GHz and Tesla P100-PCIE-16GB graphical processing unit.

4.4.2 Analysing sensitivity to the masking content

This section describes experiments that analyse if explanations from SLIME are sensitive to the content of synthetic components. For example, will SLIME explanations change if instead of occluding the interpretable components with the zero value (see Section 4.3.2), SLIME uses Gaussian noise? This analysis will assist in highlighting whether the selection of the content of synthetic components is critical in generating reliable explanations using SLIME. Moreover, the conclusions from this experiment will also apply to other local explanation methods that similarly perform input occlusion [Ribeiro et al., 2016b, Zeiler and Fergus, 2014].

SLIME aims to analyse the effect of removing a group of randomly selected interpretable components on model predictions. However, as machine listening models predict using fixed-size inputs, SLIME approximates the "removal effect" by occluding the selected interpretable components with synthetic content, hypothesising that the occluded components have minimal influence on model predictions [Fong and Vedaldi, 2017]. This section proposes five masking contents to perform input occlusion and groups them into two categories: (1) contents

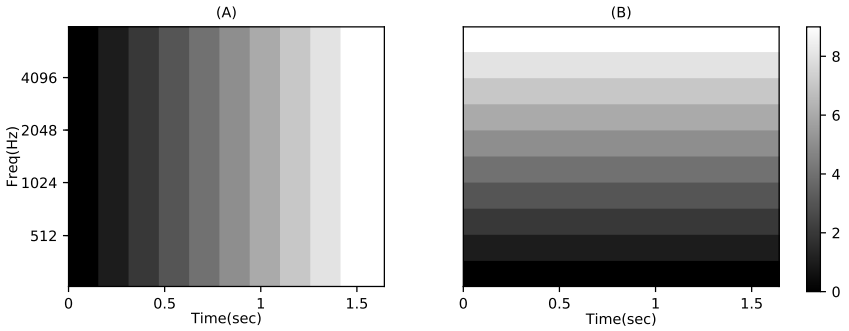

Figure 4.8: Plots depicting how SLIME performs segmentation of the mel-spectrogram representation of an audio excerpt to generate ten interpretable components. (A) Temporal segmentation and (B) spectral segmentation. The colourbar values depict the indices of the interpretable components.

that generate very low energy (near silent) components, and (2) contents that result in components with non-discriminatory audio features. The first category includes three masking contents: (1) the *zero* value, (2) the minimum bin magnitude across a dataset (min_{data}), and (3) the minimum bin magnitude in an input (min_{inp}). It is conceivable that audio feature extraction from very low energy components will result in insignificant features, which have minimal influence on model predictions. The second category includes two masking contents: (1) the average bin value in an input ($mean_{inp}$), and (2) Gaussian noise (N_g). These contents generally generate components lacking any dominant structures and thus feature extraction from such components will result in non-discriminatory audio features with limited influence on model predictions.

This section describes experiments where SLIME explains SVDNet-R1 predictions by performing input occlusion using the five different masking contents from above. The experiments aim to analyse if the modification of the content of the synthetic components changes SLIME predictions. To do that, SLIME generates top-3 explanations for eight randomly selected audio excerpts from four audio files in the Jamendo and RWC datasets. The experiment samples a vocal and a non-vocal instance from each audio file. SLIME generates temporal explanations for two excerpts from each dataset and generates spectral explanations for the remaining four audio excerpts. The experiment uses $N_s = 70000$ as it provides stable explanations for the new model (see Section 4.4.1). All other settings for executing SLIME are same as the ones for Section 4.4.1. For example, SLIME generates the temporal and spectral explanations by segmenting an input mel-spectrogram into ten interpretable components along the time and frequency axes, respectively. Fig. 4.8 depicts the temporal and spectral segmentation SLIME performs for an input audio excerpt. Table 4.3 presents

the results of the experiments.

All the explanations show the interpretable components (ICs) in the order of their influence on a prediction. The results suggest that for all the eight instances, there is little overlap in the explanations generated using different masking contents. For example, for the first six instances, there are no common ICs among the explanations generated across all the five masking contents. The explanations for those instances do include some ICs that are more frequent, but their influence in a prediction (the occurrence order) keeps changing. For example, for the instance 1888, the IC with index 6 occurs in four out of the five explanations, but its location changes from being the most influential component (the first component in the explanation) to the relatively less influential components (the second and third components in the explanation). The spectral explanations for RWC are comparatively more coherent. For example, for the instance 1578, all the three ICs and their occurrence orders are common between the explanations generated using the *zero* and $mean_{inp}$ masking contents, however, for the same instance, the explanations generated using min_{data} and N_g show similar behaviour, but includes new ICs. Similarly, the instance 12794 has one IC common among all the explanations, but with varying importance. Thus, the results suggest that for the selected audio excerpts the explanations from SLIME are sensitive to the content of the synthetic components.

The sensitivity of SLIME explanations to the content of synthetic components in the above experiment may be the result of an unsuitable N_s, as the experiment in Section 4.4.1 selects an appropriate N_s only for temporal explanations generated using the zero value as the masking content. To verify this, the experiment further analyses if $N_s = 70000$ is also an appropriate value for the other masking contents and spectral explanations, across both the datasets. Thus, for an audio excerpt sampled randomly from one of the datasets and a masking content (e.g., *zero*), the experiment applies SLIME to first generate five temporal and five spectral explanations and then calculates U_n for each explanation type by aggregating the corresponding explanations. To limit the computational time, the experiment executes the above steps for two randomly selected excerpts from each audio file in the Jamendo and RWC datasets. Thus, the experiment analyses the explanation stability for 32 and 40 randomly selected instances from the Jamendo and RWC datasets, respectively. The rest of the settings for executing SLIME are the same as in the experiment in Section 4.4.1.

Fig. 4.9 depicts the results of the above experiment. Fig. 4.9 (a) and (b) depict the distribution of U_n for the temporal and spectral explanations, respectively from Jamendo. Similarly, Fig. 4.9 (c) and (d) depict the distribution of U_n for the temporal and spectral explanations, respectively from RWC. The results

Dataset	Audio file	Index	Vocal probability	Explanation type	Explanations				
					zero	min_{data}	min_{inp}	$mean_{inp}$	N_g
Jamendo	03 - School.mp3	435	0.023	temporal	4, 3, 6	4, 1, 5	1, 3, 5	6, 9, 1	1, 0, 9
Jamendo	03 - School.mp3	3162	0.915	temporal	1, 4, 3	4, 7, 2	5, 4, 2	5, 7, 8	1, 8, 2
Jamendo	03 - Une charonge.ogg	19291	0.017	spectral	4, 6, 8	4, 5, 7	5, 1, 4	6, 8, 3	8, 5, 1
Jamendo	03 - Une charonge.ogg	1888	0.861	spectral	2, 4, 8	6, 2, 3	7, 2, 6	2, 6, 9	7, 6, 1
RWC	RWC-MDB-P-2001-M01/016 Audio Track.aiff	4732	0.233	temporal	1, 5, 0	2, 5, 4	4, 2, 7	1, 0, 8	6, 5, 4
RWC	RWC-MDB-P-2001-M01/016 Audio Track.aiff	701	0.871	temporal	5, 1, 8	2, 5, 3	5, 2, 4	1, 2, 0	6, 7, 9
RWC	RWC-MDB-P-2001-M04/4 Audio Track.aiff	1578	0.019	spectral	7, 8, 9	1, 2, 4	5, 1, 0	7, 8, 9	1, 4, 2
RWC	RWC-MDB-P-2001-M04/4 Audio Track.aiff	12794	0.966	spectral	1, 7, 8	3, 2, 1	1, 2, 6	1, 7, 8	5, 7, 1

Table 4.3: SLIME explanations for randomly selected audio excerpts from the Jamendo and RWC datasets for five masking contents. Index: instance index; Vocal probability: model confidence about the presence of singing voice; Explanations: top-3 interpretable components maximally (positively or negatively) influencing a prediction; and *zero*, min_{data}, min_{inp}, $mean_{inp}$, and N_g refer to the masking contents that occlude an input by using the zero value, minimum bin magnitude across a dataset, minimum bin magnitude in an input, average bin magnitude in an input and, Gaussian noise, respectively.

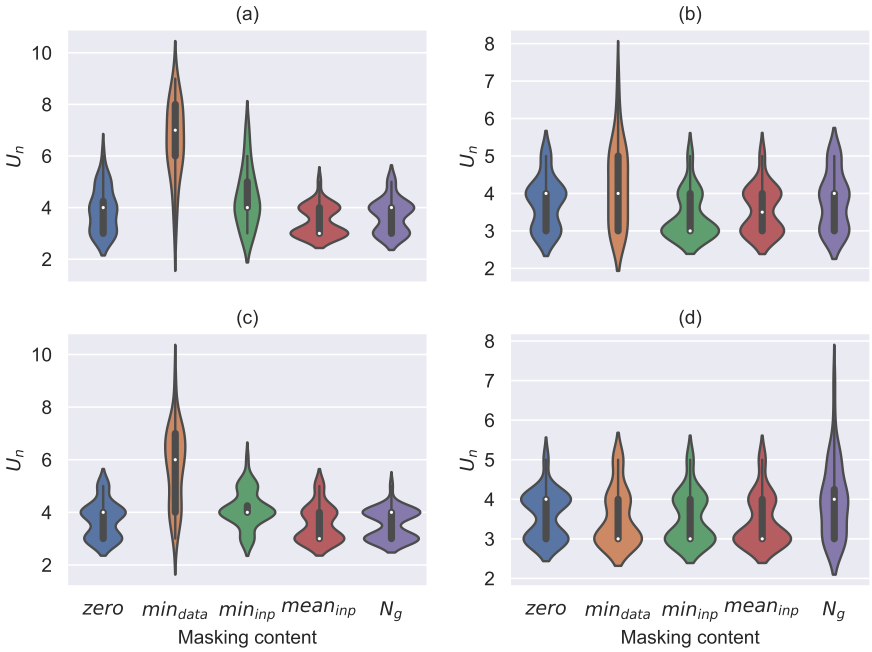

Figure 4.9: Plots depicting the influence of different masking contents on the stability of explanations from SLIME for four cases. (a) and (c) depict results for the temporal explanations from the Jamendo and RWC datasets, respectively. (b) and (d) depict results for the spectral explanations from the Jamendo and RWC datasets, respectively. U_n represents the number of unique interpretable components in explanations from applying SLIME five times to the same excerpt.

demonstrate that except for min_{data}, the explanations for the other masking contents across both the datasets have U_n distributions similar to those for the *zero* masking content. For example, Fig. 4.9 (c) suggests that for the temporal explanations from RWC, the most likely values for U_n are 4 or 3, suggesting that except for min_{data}, $N_s = 70000$ is also an appropriate value for the other masking contents. Interestingly, the explanations for min_{data} seem unstable (have high U_n values) for all except the spectral explanations for the RWC case (Fig. 4.9 (d)). This suggests that $N_s = 70000$ is not an appropriate value for this masking content and the unsuitable N_s contributed to the sensitivity of SLIME to min_{data}. Moreover, the experiment also demonstrates that for all the other masking contents, $N_s = 70000$ is an appropriate value and hence, it does not contribute to the sensitivity of SLIME explanations to the other masking contents.

The experiment also analyses the effect of masking contents on SLIME explanations on a large scale to verify whether the conclusions about the sensitivity

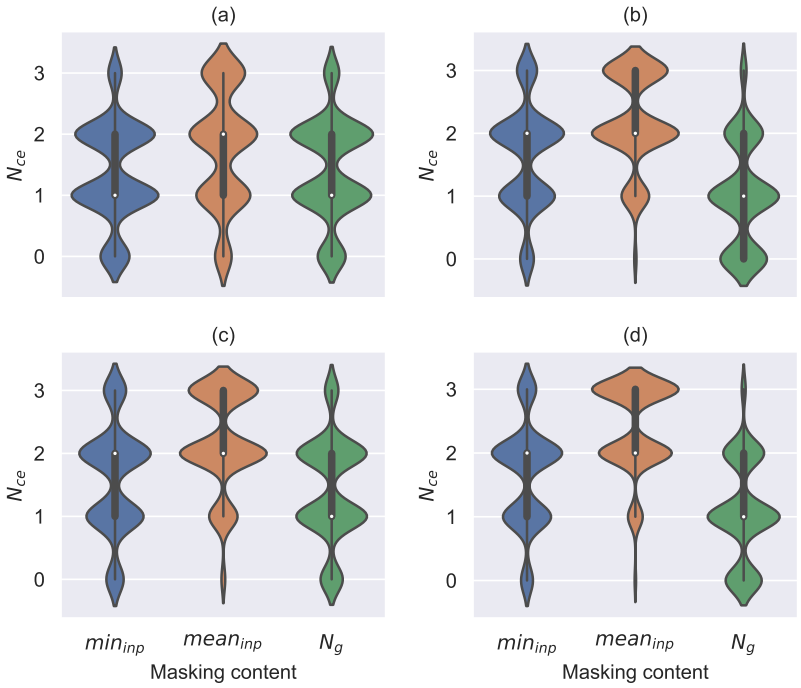

Figure 4.10: The violin plots depict the influence of different masking contents on SLIME explanations for four cases. (a) and (c) depict results for temporal explanations for the Jamendo and RWC datasets, respectively. (b) and (d) depict results for the spectral explanations for the Jamendo and RWC datasets, respectively. N_{ce} refers to the number of common interpretable components between the explanation with masking content zero and the explanation with masking content given on the horizontal axis.

of SLIME explanations to the content of synthetic components are valid for other instances. To do that, the experiment first randomly samples 50 audio excerpts from each audio file in each dataset, generating subsets with 800 and 1000 audio excerpts from the Jamendo and RWC datasets, respectively. The experiment then generates temporal and spectral explanations for the SVDNet-R1 predictions for instances in each subset, highlighting the top-3 interpretable components (positive or negative) maximally influencing the predictions. Moreover, the experiment generates the explanations for the four masking contents ($zero$, min_{inp}, $mean_{inp}$, N_g) that generated stable explanations for $N_s = 70000$. The rest of the settings for executing SLIME are the same as in the experiment in Section 4.4.1.

Fig. 4.10 depicts the results of the experiment, where subplots (a) and (b) correspond to the temporal and spectral explanations, respectively, for the Jamendo instances and subplots (c) and (d) correspond to the temporal and spectral explanations, respectively, for the RWC instances. Each plot depicts the

distribution of the number of common interpretable components (N_{ce}) between explanations generated using two different masking contents, one of which is the reference. This experiment uses the masking content *zero* as the reference and thus, for each explanation, the experiment computes how many components remain the same when SLIME replaces the reference masking content with the other masking contents (e.g., min_{inp}).

The analysis of the above results provides interesting insights into the behaviour of SLIME. For example, the explanations generated using $mean_{inp}$ are closer to those generated using the reference with low likelihood of no overlap in explanations, although for the temporal explanations for the Jamendo instances case, there exists a fair likelihood of $N_{ce} = 1$. On the other hand, the explanations generated using the other masking contents have comparatively lower overlap with the explanations using the reference masking content. Moreover, for N_g, the likelihood of no overlap in interpretable components is fairly high. Overall, the results demonstrate that for both the explanation types and datasets, the explanations from SLIME are sensitive to the content of synthetic components. This suggests that selecting an appropriate masking content is crucial for generating reliable predictions from SLIME. The next section discusses a method that assists in identifying the appropriate masking content using audio stem files.

4.4.3 Generating reliable explanations

The previous section demonstrated that SLIME explanations are sensitive to the content of the synthetic components. Thus, the generation of reliable explanations from SLIME requires a careful selection of the content of the synthetic components. This section proposes to select an appropriate masking content for the local explanations from SLIME in two steps. Step 1 involves using domain knowledge to select a list of relevant masking contents, as a content suitable for one domain or explanation type may not be suitable for the other. For example, the zero value may be a suitable masking content for an RGB image with pixel values between 0 (lowest intensity) to 255 (highest intensity), but may not be suitable for a log-scaled spectrogram with values between −min to +max as it may make quieter sections louder. Step 2 involves using the ground-truth annotations in selecting an appropriate masking content from the list of suitable contents. For example, for the SVD models in this chapter, the ground-truth indicates which temporal segments in an input audio contain vocals. SLIME can use this information to select an appropriate masking content for the temporal explanations by first generating the temporal explanations using all the suitable masking contents and then selecting the content that generates the temporal

explanations with maximum overlap with the ground-truth. For example, the ground-truth annotations for an input audio denote that temporal segments 2 and 4 contain the singing voice and the remaining temporal segments (1, 3, and 5) contain non-vocals. An SVD model predicts that this input contains vocals with a confidence score of 0.95. The temporal explanations from SLIME that highlight the top-2 super-samples positively influencing the prediction for each of the three suitable masking contents 'a', 'b', and 'c' are $\{1, 2\}$, $\{2, 4\}$, and $\{3, 5\}$, respectively. The super-samples are the output of the input segmentation at the ground-truth boundaries. This suggests that the masking content 'b' is appropriate as it generates explanations that completely overlap with the ground-truth.

This section demonstrates the above method for the temporal explanations of the predictions of the SVDNet-R1 model. However, instead of using audio excerpts and the ground-truth annotations from the Jamendo and RWC datasets, the experiment used synthesised audio excerpts and their corresponding ground-truth annotations. This approach has two main benefits. First, it allows to synthesise a large dataset for experimentation which is not possible with the existing datasets as due to the short duration of the model inputs (around 1.6 secs), the majority of audio excerpts contain either vocals or non-vocals, thus, are not useful for the experiment. Second, the ground-truth annotations for the Jamendo and RWC datasets may be noisy and thus, using the synthesised dataset assists in performing more controlled experimentation.

The experiment generates the synthetic dataset using the ccMixter dataset that includes a vocal and a non-vocal stem for each of the 50 songs it contains (see Chapter 2). The dataset synthesis involves first randomly selecting ten mel-spectrogram excerpts from each stem file corresponding to a song. Each mel-spectrogram excerpt is around 1.6 seconds. In this experiment, the temporal indices of the selected mel-spectrograms are the same for the vocal and non-vocal stems. For example, the method extracts a mel-spectrogram excerpt at time index = 20 seconds both from the vocal and non-vocal stems. Thus, the method samples ten pairs of mel-spectrograms, where each pair contains mel-spectrograms belonging to the vocal and non-vocal stems and sampled at the same time index. The method segments each mel-spectrogram in a pair into ten temporal segments as in Section 4.4.2. The method randomly selects three temporal segments from the vocal stem and replaces the corresponding temporal segments in the non-vocal stem using them. The method repeats this process four times, generating four mel-spectrograms, where each mel-spectrogram contains seven non-vocal segments and three vocal segments. Thus, for each song in the dataset, the method generates 40 mel-spectrograms and executing the method on the whole dataset generates 2000 mel-spectrograms and their anno-

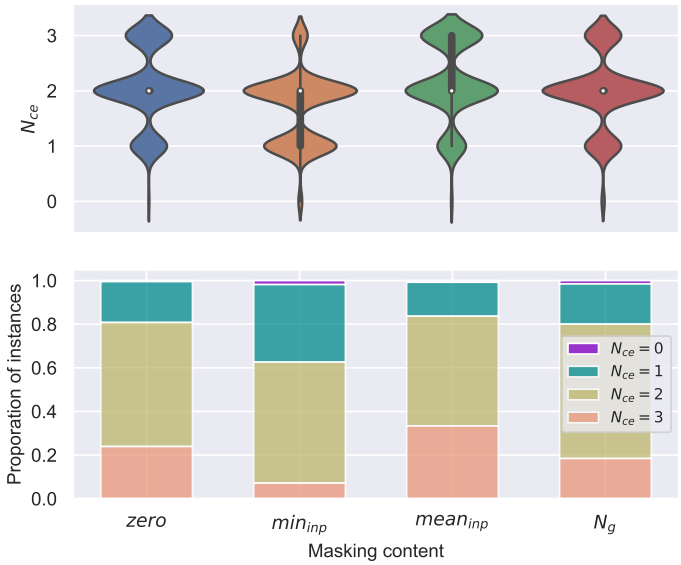

Figure 4.11: Plots depicting the influence of different masking contents on temporal explanations from SLIME for randomly selected instances from the synthetic dataset. The top plot depicts the distribution of the number of common interpretable components N_{ce} between the ground-truth explanation and the temporal explanation generated with the masking content given on the horizontal axis. The bottom plot depicts the proportion of instances (audio excerpts) for different N_{ce} values corresponding to all the four masking contents.

tations that indicate the temporal segments containing vocals. This experiment aims to analyse only the true positive instances, i.e., those excerpts that contain vocals and are correctly predicted by the SVDNet-R1 model. Thus, the final dataset the experiment uses for selecting an appropriate masking content has 656 excerpts.

The experiment uses SLIME to generate the temporal explanations for the prediction of each excerpt in the synthetic dataset. The temporal explanations highlight the top-3 interpretable components that positively influence SVDNet-R1 predictions. The experiment generates the temporal explanations using the four masking contents (*zero*, min_{inp}, $mean_{inp}$, N_g) that resulted in stable explanations for $N_s = 70000$. The rest of the arguments for executing SLIME are the same as in Section 4.4.1. The experiment generates four temporal explanations, one corresponding to each masking content, for each audio excerpt in the synthetic dataset. The experiment selects an appropriate masking content by computing the number of common interpretable components (N_{ce}) between the ground-truth explanations and the temporal explanations for each masking content.

Fig. 4.11 depicts the results of the experiment. Fig. 4.11 (top) depicts the distribution of N_{ce} corresponding to each masking content. Fig. 4.11 (bottom) presents the proportion of audio excerpts for each N_{ce} corresponding to each masking content.

The results provide useful insights into the behaviour of SLIME for different masking contents, assisting in the selection of an appropriate content. The results suggest that for around 34% of the instances, the temporal explanations corresponding to $mean_{inp}$ completely match the ground-truth. The indices of the interpretable components in the explanations are the same, however, their order of occurrence may differ. For the masking contents *zero*, min_{inp}, and N_g, this number is 23.9%, 7.16%, and 18.44%, respectively. This suggests that among all the masking contents used in this experiment, $mean_{inp}$ generates the most accurate temporal explanations. The accuracy of SLIME seems low even for the best masking content, however, it is important to note that SLIME learns explanations using model predictions to the perturbed versions of an input. Thus, less accurate model predictions will result in less accurate SLIME explanations. In this experiment, the predictions from SVDNet-R1 will be less accurate for the synthetic instances as the model was not trained on the ccMixter dataset, and thus, the synthetic samples are out-of-distribution samples. An example of the difference in the training and test distributions is the composition of the vocal category. In the Jamendo dataset used to train the SVDNetR1 model, the vocal category contains both the vocal and non-vocal sounds, however, in the synthetic dataset, the vocal category contains only the singing voice. Thus, using the same data distribution may result in more accurate explanations from SLIME that completely match the ground-truth.

The results also show that the % of instances with temporal explanations having at least two interpretable components in common with the ground truth is 80.79%, 62.65%, 83.68%, and 80.03% for the masking contents *zero*, min_{inp}, $mean_{inp}$, and N_g, respectively. This suggests that *zero*, $mean_{inp}$, and N_g perform similarly in reliably identifying at least two interpretable components positively influencing a prediction. Moreover, the % of instances where SLIME explanations have no overlap with the ground-truth is very low ($< 2\%$) across all the masking contents.

The experimental results suggest that among the four masking contents, $mean_{inp}$ is the most accurate in generating temporal explanations that perfectly match the ground-truth. This suggests that the SVDNet-R1 model is more sensitive to the other masking contents [Mittelstadt et al., 2019], and hence, the hypothesis that occluding input components with the other masking contents is equivalent to removing the corresponding input components, seems weaker for those contents. The results further show that except for min_{inp}, temporal

explanations generated using the other masking contents have at least two input components overlapping with the ground-truth for around 80% of instances.

4.5 Summary and conclusion

This chapter proposed SLIME, a local explanation generation method that extends the applicability of LIME [Ribeiro et al., 2016b] to machine listening. The chapter highlighted the need for interpretable explanations in analysing machine listening models and discussed how SLIME generates such explanations using three different interpretable representations (temporal, spectral and time-frequency) of an input.

The chapter described two experiments that demonstrated the effectiveness of SLIME in understanding machine listening models. The first experiment used SLIME to analyse the behaviour of two shallow SVD models. The experiment generated temporal explanations for specific instances from the Jamendo dataset, highlighting super-samples maximally influencing their predictions. The results suggested that the explanations help reveal how the BDT model is making decisions based on content that does not contain singing voice despite possessing high classification accuracy for the selected instances. Such issues cast doubt on the generalisability of the model. The results also suggested that the RF model seems more trustworthy as the super-samples maximally influencing its predictions for the selected instances contain vocals. The second experiment used time-frequency explanations from SLIME to validate the behaviour of the state-of-the-art deep SVD model. The analysis of the time-frequency explanations for a specific instance suggested that the model seems trustworthy. The experiment also compared the time-frequency explanations from SLIME with saliency maps. The results suggested that there exists a fair agreement in the explanations from both the methods.

The chapter then described two experiments that analysed the robustness of SLIME to changes in the values of two input parameters. The first experiment analysed if the SLIME explanations are sensitive to the number of samples (N_s) SLIME generates in the interpretable space. The results demonstrated that the SLIME explanations are unstable for lower values of N_s, however, for the higher values of N_s, SLIME generates stable explanations. The experiment also demonstrated that sampling a larger number of synthetic samples will linearly increase the explanation generation time of SLIME. This suggests that the selection of an appropriate value for N_s should also consider the explanation generation time from SLIME.

The second experiment analysed if SLIME explanations are sensitive to how SLIME occludes inputs to generate synthetic samples. SLIME randomly oc-

cludes input components by replacing their content with synthetic content. This experiment analysed if SLIME needs to carefully select the masking content. To do this, the experiment analysed the temporal and spectral explanations corresponding to four proposed masking contents for instances from two different datasets. The results suggested that SLIME explanations are sensitive to the masking content and its careful selection is crucial in generating reliable explanations. This is an important result as it suggests that any explanation method that uses input occlusion in its explanation generation pipeline may also be sensitive to the masking content.

Finally, the chapter described a method to use the ground-truth annotations for selecting an appropriate masking content for temporal explanations from SLIME. An experiment demonstrated the proposed method for instances from a synthetic dataset. The results suggested that the average bin magnitude of an input seemed to be an appropriate masking content for temporal explanations as it generates explanations having at least two interpretable components matching with the ground-truth for around 84% of the instances. Thus, the experiments in this chapter suggest that SLIME is an effective method to analyse the local behaviour of deep and shallow machine listening models.

It is important to note that this chapter used different models in different experiments. Specifically, the experiments used the two shallow models for temporal explanations (Section 4.3.1), SVDNet for time-frequency explanations (Section 4.3.2) and preliminary analysis of N_s (Section 4.3.3), and SVDNet-R1 for all the other experiments (section 4.4). This happened mainly due to differences in the motivations behind the experiments. The experiments in Sections 4.3.1 and 4.3.2 aimed to demonstrate that SLIME can be used to verify the behaviour of shallow and deep machine listening models. The deep model SVDNet-R1 was not used for the experiments in Section 4.3.2 because the model was unavailable when the experiments were performed. Due to the discontinuation of the Theano framework in 2017, this Tensorflow-based model was trained for experiments in Chapter 5 and later used for all the experiments in Section 4.4.

The analysis of instance-based explanations highlighted some challenges associated with SLIME (e.g., sensitivity to parameters), hence, all the other experiments focussed on understanding the behaviour of SLIME to changes in parameters. These experiments used only the deep models as they are complex and state-of-the-art in SVD. The analysis of the robustness of SLIME to shallow models is a part of the proposed future work (see section 7.2). Moreover, SVDNet-R1 was not used for experiments in Section 4.3.3 due to its unavailability and SVDNet was not used for experiments in Section 4.4 as it has the same architecture and very similar performance (see Chapter 3) to that of SVDNet-

R1.

4.6 Reproducibility

The code for all the experiments is open-sourced. The below-mentioned Github repositories contain the code and parameters to reproduce the thesis results, and steps to generate new results.

- Experiments in Section 4.3. This includes generating the temporal and time-frequency explanations for the predictions of the SVD models - `https://github.com/saum25/SoundLIME`
- Experiments in Section 4.4. This includes the experiments to analyse the robustness of SLIME and the synthetic dataset and corresponding ground-truth annotations to generate reliable predictions using SLIME - `https://github.com/saum25/local_exp_robustness`

Chapter 5

Activation maximisation

This chapter focuses on analysing the global behaviour of machine listening models. Specifically, this chapter discusses *activation maximisation* (AM), a global analysis method that synthesises examples in the input space (e.g., images, spectrograms) to maximally activate components of a DNN (e.g., a neuron, a layer). This chapter introduces novel methods for AM and for identifying suitable AM hyper-parameters that generate interpretable examples. Moreover, the chapter demonstrates the effectiveness of the proposed methods by using them to understand the concepts that two deep SVD models learn in their output layers.

This chapter consolidates the work from a workshop paper [Mishra et al., 2019] that proposed to use a *generative adversarial network* (GAN) [Goodfellow et al., 2014] as a prior in the AM pipeline and introduced a new metric that uses the *Fréchet inception distance* (FID) [Heusel et al., 2017] to select suitable AM hyper-parameters. The paper demonstrated the proposed methods for a state-of-the-art deep SVD model (SVDNet-R1, see Section 3.4.5). In addition to the content from the workshop publication, this chapter includes further experiments to analyse the global behaviour of the single output layer neuron in SVDNet-R1. Moreover, the chapter also includes a new section that describes experiments to apply the proposed AM methods to analyse a deep SVD model (SVDNet-R2, see Section 3.4.5) that is architecturally similar to SVDNet-R1, but for its output layer that contains two neurons.

The remainder of this chapter is organised as follows: Section 5.1 discusses the existing challenges in using AM to understand a DNN and highlights our contributions that aim to address those challenges. Section 5.2 describes our proposed methods for performing AM and for automatically selecting suitable hyper-parameters that assist in the generation of interpretable examples. Section 5.3 describes experiments that use the proposed methods to understand the

concepts that two deep SVD models learn in their output layer neurons. Section 5.4 describes the results of the experiments. Section 5.5 presents a perceptual study to analyse some observations from the qualitative analysis in earlier sections. Section 5.6 summarises the key results and highlights the effectiveness of the proposed methods in globally understanding DNNs. Finally, Section 5.7 mentions the repositories that host the code, trained models, and synthesised examples for all the experiments in this chapter.

5.1 Introduction

Chapter 4 introduced SLIME, which helps in understanding the behaviour of a machine listening model by explaining its predictions. The analysis of model predictions provides useful insights into model behaviour and helps validate model trustworthiness. However, the local analysis focuses on why a model takes a specific decision and provides limited insight into the process by which a model forms its predictions. The understanding of the functioning and interactions of model components may assist in improving model performance and in verifying its trustworthiness [Yosinski et al., 2015, Zeiler and Fergus, 2014].

We can analyse components of a DNN using different methodologies (see Chapter 2). For example, one can use feature inversion to map latent codes to the input space highlighting the discriminative information a DNN preserves at its layers [Dosovitskiy and Brox, 2016a, Mahendran and Vedaldi, 2015]. Chapter 6 describes experiments that use this method to understand a deep machine listening model. In another direction, one can analyse features that different components of a DNN are sensitive to. One way to do this is by using *data-driven AM* that identifies dataset instances that maximally activate different components in a DNN [Zhou et al., 2015]. Another way is by using *vanilla AM* [Erhan et al., 2009] that iteratively optimises synthetic examples initialised with random noise in the input space (e.g., images) to maximally activate a neuron or layer in a DNN. Since vanilla AM is data-independent and tends to focus more on the explanatory input factors, this chapter will pursue a vanilla AM-based approach for model analysis. It is important to note that the terms vanilla AM and data-driven AM do not originate from the literature, however, to differentiate between the two ways of performing AM, this thesis introduces these terms.

The interpretability of examples generated by vanilla AM depends on two key factors: optimisation of hyper-parameters and the prior. Generally, interpretable examples are selected for each neuron by performing a grid search in the hyper-parameter space and visually inspecting each generated example [Nguyen et al., 2016a], but this is subjective, prohibitively slow and limits the hyper-

parameter search space. Also, such an approach is not scalable to analysing other DNN neurons that may require different hyper-parameter settings.

The use of priors for vanilla AM restricts the input search space to prevent generating uninformative, adversarial examples. Researchers have proposed several hand-crafted priors for effective vanilla AM, synthesising interpretable images [Yosinski et al., 2015, Nguyen et al., 2016b, Mahendran and Vedaldi, 2015]. In another direction, [Nguyen et al., 2016a] demonstrated that replacing hand-crafted priors by a learned prior (adversarially trained feature inverter) considerably improves the interpretability of synthesised images. However, their approach requires training a separate prior for each layer in the classifier, and appears to rely on the prior and the classifier model having similar architectures.

This chapter aims to tackle these challenges, making the following contributions:

- To our knowledge, this chapter is the first to use a GAN for example generation using vanilla AM. The GAN imposes a strong prior and enables effective AM for any given part of a classifier and even other classifiers with the same input domain without re-training the generator. The work by Nguyen et al. [2017] is closest to the proposed approach, in which the authors use a denoising autoencoder as a prior on the latent code of the adversarially trained feature inverter.

- This chapter proposes a quantitative measure estimating the interpretability of a set of generated examples by adopting the Fréchet inception distance [Heusel et al., 2017]. The chapter provides evidence for its effectiveness by qualitatively analysing the synthesised examples.

- The chapter applies the proposed methods to two machine listening models. One of the models is a state-of-the-art deep audio classification model that predicts singing voice activity in music excerpts (this chapter refers to this model as SVDNet-R1) and the other model is a variant of SVDNet-R1 and has two neurons in its output layer. The experiments generate visualisations that successfully capture the concepts represented by the ground truth labels the models were trained to predict. There have been some recent works in the global analysis of deep audio classification models, but they either use a different method [Mishra et al., 2018b] or perform vanilla AM with hand-crafted priors [Zhang and Duan, 2018].

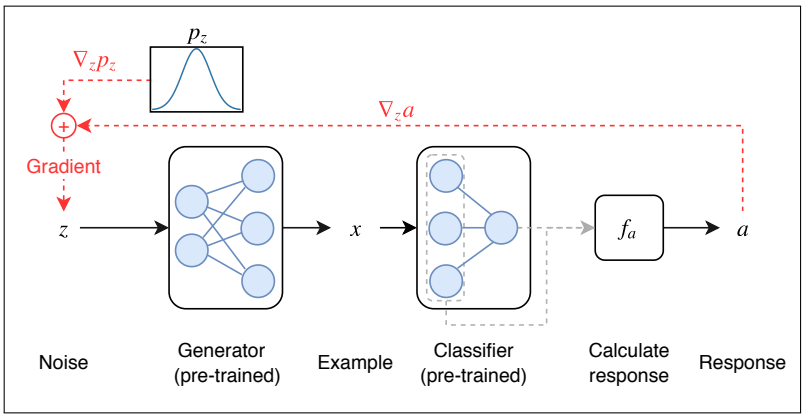

Figure 5.1: Overview of the proposed approach for performing vanilla activation maximisation. A noise vector z is used to generate an example x, for which a response $a \in \mathbb{R}$ is calculated with a response function f_a from all neuron activations of the classifier. f_a can be defined depending on which aspect of the classifier is of interest; examples include the activation of a certain neuron, or the average layer activation. z is optimised to maximise the response a, but also the prior probability $p_z(z)$ to favour realistic outputs.

5.2 Method

Figure 5.1 provides an overview of the proposed method for performing vanilla activation maximisation. For a pre-trained neural network classifier[1] f_c with M neurons and input $x \in \mathbb{R}^d$, the goal of the method is to synthesise examples that activate a given *neuron activation pattern* ("classifier response"). Formally, the method defines $f_n(x) \in \mathbb{R}^M$ as the output activations of *all* M neurons in the classifier f_c for a given input example x. The classifier response the method aims to explain can then be defined in a general fashion as the output of some function $f_a : \mathbb{R}^M \to \mathbb{R}$ that takes *all* M neuron activations of the classifier as input. f_a can be set to output the activation of a single neuron, or the average activation of one or multiple layers, but any differentiable function is supported.

5.2.1 Vanilla activation maximisation

We can perform vanilla activation maximisation to find an input example $\hat{x} \in \mathbb{R}^d$ so that the resulting activation $f_a(\cdot)$ is maximised:

$$\hat{x} = \arg\max_x f_a(f_n(x)) \tag{5.1}$$

[1]The models used for experiments in this chapter are classification models, hence in this chapter, the terms 'model' and 'classifier' are used interchangeably.

Algorithm 2: Activation maximisation using a GAN as a prior

Input: Pre-trained classifier f_c with the all neuron activation function f_n and a response function f_a
Input: Pre-trained GAN with generator f_g
Input: The number of iterations N_t, the regularisation parameter λ, Adam hyper-parameters - initial learning rate l_r, β_1, β_2, ϵ
Output: Maximally activating input $\hat{x}$

Sample a k-dimensional noise vector $z \sim \mathcal{N}(\mathbf{0_k}, \mathbf{I_k})$;
for $i \in \{1, 2, 3, ..., N_t\}$ **do**
    $a \leftarrow f_a(f_n(f_g(z)))$; `// compute classifier response`
    $C \leftarrow a + \lambda \log p_z(z)$;
    $z \leftarrow \text{Adam}(\nabla_z C, z, l_r, \beta_1, \beta_2, \epsilon)$;
end for
$\hat{x} \leftarrow f_g(z)$;
return $\hat{x}$;

We can optimise the above objective using stochastic gradient descent (SGD) by backpropagating through the classifier layers (see Chapter 2).

5.2.2 GAN-based prior

Vanilla activation maximisation often produces adversarial examples [Nguyen et al., 2015], which can be very different from inputs encountered during classifier training and testing, are hard to interpret and do not explain the classifier's behaviour for real-world inputs. Furthermore, optimising over the input x directly is often difficult, especially if the dimensionality d is high [Nguyen et al., 2016a].

The proposed method makes use of a GAN (for more details, see Goodfellow et al. [2014] and Section 2.1.3), where a generator $f_g : \mathbb{R}^k \to \mathbb{R}^d$ is trained to map a noise vector $z \in \mathbb{R}^k$ drawn from a known noise distribution p_z to a generated example x, and optimises

$$\hat{z} = \arg\max_z (f_a(f_n(f_g(z))) + \lambda \log p_z(z)). \tag{5.2}$$

The weighting term $\lambda \geq 0$ is a hyper-parameter controlling the trade-off between vanilla activation maximisation and the realism of the generated examples. This chapter refers to λ as *regularisation parameter*. Note that the method searches in the low-dimensional noise space for a vector $\hat{z}$ whose associated generator output $f_g(\hat{z})$ produces a high activation, which avoids optimisation issues. To encourage realistic outputs, the real data density p_x should ideally be used in the form of a prior term $\log p_x(f_g(z))$ in (5.2), but the method does not have access to p_x. However, assuming a well-trained generator, the method can use

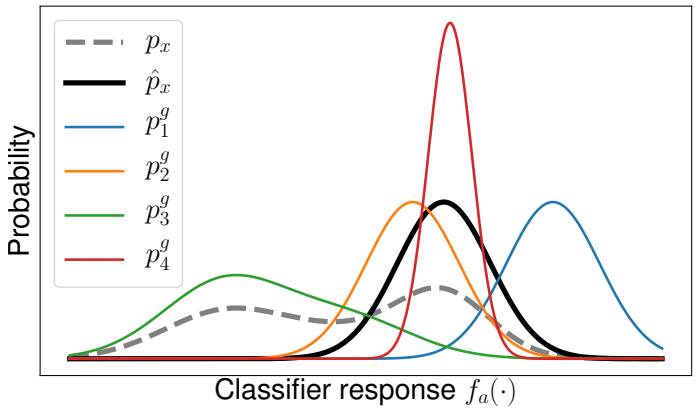

Figure 5.2: Intuitive explanation for the proposed metric, showing the distributions of activations $f_a(\cdot)$ obtained for input examples from the dataset (p_x), of the dataset examples with the highest N responses $f_a(\cdot)$ ($\hat{p}_x$), and of four hypothetical generators, $p_1^g, \ldots . p_4^g$. The proposed metric determines which generator distribution is most similar to $\hat{p}_x$ to ensure realistic examples.

$\log p_z(z)$ instead, since it should be approximately proportional.

To optimise (5.2) with gradient descent, the method requires p_z to be a continuously differentiable distribution. Note that this does not include the uniform distribution commonly used for training GANs, for example in [Goodfellow et al., 2014, Radford et al., 2016, Hjelm et al., 2017]. Algorithm 2 presents the pseudocode of the proposed activation maximisation method.

5.2.3 Example generation

The previous section 5.2.2 demonstrated how one example is generated by the proposed approach. To generate a set of N examples, the method draws N random noise vectors $\tilde{z}_1, \ldots, \tilde{z}_N$ independently from p_z as initialisation points for SGD. The resulting examples should be diverse, so converging to the same optima of (5.2) independent of initialisation is undesirable. Therefore the method sets the SGD *learning rate* l_r as well as the *number of update steps* N_t as hyper-parameters, since they control the influence of the initialisation points on the generated examples and thereby the amount of diversity and randomness.

5.2.4 Hyper-parameter optimisation

To optimise the regularisation parameter λ and optimisation parameters l_r as well as N_t, it would be ideal to have human subjects evaluate the usefulness of the explanations resulting from different configurations, but this is prohibitively time-intensive. Here usefulness is used in the context of understanding model behaviour. In other words, are explanations synthesised by hyper-parameter

configurations interpretable so as to help understand the global behaviour of a model. Therefore, this chapter introduces a metric for quickly evaluating a set of generated explanations, allowing efficient hyper-parameter optimisation. The following content explains the reasoning of the proposed method using the hypothetical example in Figure 5.2. The method posits that good interpretability requires the generated examples to have a similar distribution of classifier responses $f_a(\cdot)$ as the N samples with the highest response from the dataset ($\hat{p}_x$ in Figure 5.2). This is because unrealistic adversarial examples (generator 1 in Figure 5.2) often lead to large responses. Also, too much weight on the GAN prior λ or ineffective optimisation (generator 3) leads to examples that are realistic, but have too low responses compared to real examples. Additionally, the variance of responses should be similar (making generator 2 the best according to the proposed metric) to ensure a sufficient degree of diversity in the generated samples (in contrast to generator 4).

To take the average and the variance of the responses into account, the proposed method adopts the Fréchet Inception Distance (FID) [Heusel et al., 2017] as the distance metric (see Section 2.1.3). Since the responses f_a for each example are scalar values, the FID reduces to

$$d^2((\mu_r, \sigma_r), (\mu_g, \sigma_g)) = (\mu_r - \mu_g)^2 + \sigma_r + \sigma_g - 2(\sigma_r \sigma_g)^{\frac{1}{2}} \tag{5.3}$$

where μ_r and μ_g are the means and σ_r and σ_g the unbiased sample variance of the (one-dimensional) real and the generated response distribution, respectively.

The following section 5.3 will qualitatively investigate whether the metric proposed above adequately reflects the interpretability of a set of generated examples.

5.3 Experiments

To analyse the effectiveness of the proposed methods (GAN-based vanilla AM and automatic selection of suitable hyper-parameters) and to investigate whether approaches based on vanilla AM can transfer to domains other than computer vision, this chapter applies the proposed methods to machine listening models. Specifically, this chapter will consider singing voice detection, a binary classification task [Lee et al., 2018] where a classifier predicts whether singing voice is present in a segment of a music recording (see Chapter 2 and Chapter 3).

5.3.1 Choice of machine listening models

This chapter selects a state-of-the-art SVD model introduced by Schlüter and Grill [2015] for experiments. The model (referred to as SVDNet in this thesis) is an eight-layer CNN that takes a mel-spectrogram of an audio excerpt of around 1.6s duration as input. Using a single neuron with sigmoid activation in the last layer, the CNN predicts the probability of singing voice being present at the centre of the input audio excerpt. Chapter 3 provides the architecture, training details and performance evaluation of SVDNet.

The experiments apply the proposed methods to a replicated version of SVDNet, referred in this thesis as SVDNet-R1. As mentioned in Chapter 3, the performance of SVDNet-R1 is very similar to the one reported by the authors for SVDNet. Additionally, the experiments apply the proposed methods to a variant of SVDNet-R1. The variant (referred to as SVDNet-R2 in this thesis) has two neurons in the output layer, each corresponding to one of the classification categories. The softmax layer transforms the output scores of the two neurons to the classification probabilities. The rest of the architecture, training and evaluation procedure for SVDNet-R2 remains the same as for SVDNet-R1. Chapter 3 mentions the performance of SVDNet-R2 on the Jamendo dataset. The results suggest that the model performs comparably to SVDNet-R1.

5.3.2 Choice of response function

This chapter focuses on analysing the features that the output layer neurons are sensitive to in both the machine listening models. Similar research in computer vision has discovered that such neurons learn high-level concepts representative of their classification categories [Nguyen et al., 2016a, Zeiler and Fergus, 2014]. Thus, for SVDNet-R1, the experiments generate positive and negative examples that maximally and minimally excite its output layer neuron, respectively. Compared to using other definitions of f_a, this allows us to directly evaluate the characteristics of our generated examples, as the positive examples should differ from the negative ones by the presence of singing voice since the model is known to be accurate at singing voice detection. Similarly, for SVDNet-R2, the experiment generates examples that maximally activate each neuron in its output layer and analyses whether examples represent high-level class characteristics.

The initial experiments for the SVDNet-R1 model show that maximising or minimising the predicted probability (post-sigmoid activation for the output neuron) converges to 0 or 1 after only very few iterations, leading to vanishing gradients due to saturation of the sigmoid non-linearity, effectively halting optimisation. This indicates an inherent problem of the classifier and not of the proposed method, as neural networks are well-known to be prone to mak-

Layer	Input shape	Filter size	Stride	No. of filters/neurons	Output shape
FC	128	-	-	5120	5120
ConvT	$8 \times 5 \times 128$	5×10	2×2	64	$16 \times 10 \times 64$
ConvT	$16 \times 10 \times 64$	5×20	2×2	32	$32 \times 20 \times 32$
ConvT	$32 \times 20 \times 32$	5×20	2×2	16	$64 \times 40 \times 16$
ConvT	$64 \times 40 \times 16$	5×20	2×2	8	$128 \times 80 \times 8$
Conv	$128 \times 80 \times 8$	5×5	1×1	1	$128 \times 80 \times 1$

Table 5.1: The architecture of the GAN generator. Input and output shapes are ordered as: time × frequency × number of channels. FC, ConvT, and Conv refer to the fully-connected, transposed convolutional and convolutional layers, respectively.

ing over-confident predictions [Gal and Ghahramani, 2016]. Moreover, earlier research has shown that maximally activating softmax outputs results in examples with poor interpretability [Simonyan et al., 2014]. Thus, for both the models, the experiments maximise and minimise the pre-sigmoid or pre-softmax activations of the final layer neurons instead.

5.3.3 GAN training

The experiment uses the free music archive (FMA) dataset [Defferrard et al., 2017] for training the GAN (see Section 2.1.3). The FMA dataset is a collection of 106,574 full-length tracks and metadata, available under the creative commons license extracted from the free music archive library[2]. The musical audio files are stereo, sampled at 44.1 kHz, and encoded using the mp3 standard. The tracks are from 16,341 artists and 14,854 albums and arranged in a hierarchical taxonomy of 161 genres (16 main genres and 145 sub-genres). The experiment trains the GAN by using only Pop music pieces (around 18,000 full-length tracks) to reduce the data complexity and to make the song selection more similar to the one used for training the machine listening models.

The experiment trains the GAN using mel-spectrogram excerpts of around 1.6 seconds duration. The experiment extracts the excerpts from the audio tracks of the FMA dataset using the pre-processing steps from training the SVDNet model (see Section 3.4.1, Chapter 3).

For the GAN generator, the experiment chooses a standard normal likelihood $\mathcal{N}(z|\mathbf{0}_n; \mathbf{I}_n)$ for the continuously differentiable noise term $p_z(z)$, with a dimensionality of $n = 128$. The generator is a CNN with architecture adapted from the DCGAN [Radford et al., 2016] and is shown in Table 5.1. Thus, there are no max-pooling layers, instead, following DCGAN, the generator uses multiple strided transposed convolutions with the same 2×2 stride. This upsamples the input by a factor of 2 after each transposed convolutional layer. Moreover,

[2]https://freemusicarchive.org

Layer	Input shape	Filter size	Stride	No. of filters/neurons	Output shape
Conv	$128 \times 80 \times 1$	5×80	2×2	32	$64 \times 40 \times 32$
Conv	$64 \times 40 \times 32$	5×40	2×2	64	$32 \times 20 \times 64$
Conv	$32 \times 20 \times 64$	5×20	2×2	128	$16 \times 10 \times 128$
Conv	$16 \times 10 \times 128$	5×10	2×2	256	$8 \times 5 \times 256$
FC	10240	-	-	1	1

Table 5.2: The architecture of the GAN discriminator. Input and output shapes are ordered as: time × frequency × number of channels. Conv and FC refer to the convolutional and fully-connected layers, respectively.

similar to the DCGAN generator, the number of feature maps at each layer decreases by a factor of 2, but the two architectures differ in the number of feature maps at each layer. Finally, the GAN generator used in this chapter uses non-uniform filters that differ from the uniform 5×5 filters used by the DCGAN generator. The transposed convolutional layers, as well as the fully-connected layer, in the GAN generator, have leaky-ReLU non-linearity with scaling factor = 0.2 [Mass et al., 2013].

The final convolution layer in the GAN generator outputs a $128 \times 80 \times 1$ tensor, which is cropped evenly at the borders to obtain 115 time frames as required by the machine listening models. The final convolution employs $x \rightarrow \max(x, \log(10^{-7}))$ as activation function to ensure the generated spectrogram magnitudes are in the same interval range as the mel-spectrograms obtained from preprocessing real audio samples, where x is the bin magnitude. It is important to note that the DCGAN generator does not use any convolutional layer, does not perform any output clipping, and uses the ReLU non-linearity in all but the last layer where it uses the tanh non-linearity.

The discriminator architecture is again similar to the DCGAN [Radford et al., 2016] and is shown in Table 5.2. The discriminator makes use of multiple strided 2D convolutions with 2×2 stride to process the mel-spectrogram input of size $115 \times 80 \times 1$. The output is a scalar real value used to distinguish real from generated samples. The convolutional layers have leaky-ReLU activations (scaling factor = 0.2) and bias. The fully-connected layer has no bias or activation function. The number of feature maps at each convolutional layer increases by a factor of 2. Importantly, the GAN discriminator in this chapter differs from the DCGAN discriminator in the number of feature maps at each layer and in the use of non-uniform filters that convolve across the full frequency axis. The DCGAN discriminator uses uniform 5×5 filters.

The experiment uses the WGAN-GP (Wasserstein GAN with gradient penalty) objective for training the GAN as in [Gulrajani et al., 2017], with a gradient penalty weight of 10 (see Section 2.1.3). The training process uses the Adam optimiser [Kingma and Ba, 2015] with a learning rate of 10^{-4} to train the gen-

erator and discriminator for 600,000 iterations with a batch size of 16.

5.3.4 AM Optimisation

The vanilla AM approach this chapter proposes involves three hyper-parameters: the initial learning rate l_r, the number of iterations N_t, and the prior weight (regularisation parameter) λ (see Section 5.2.4). The experiments perform a grid search over the hyper-parameter space, using $l_r \in \{0.1, 0.01, 0.001\}$, $\lambda \in \{0.1, 0.01, 0.001\}$ and $N_t \in \{100, 500, 1000\}$, giving 27 possible settings of the hyper-parameters. The experiments sample $N = 50$ noise vectors from the noise distribution p_z, resulting in $N = 50$ examples along with their respective activation values $f_a(\cdot)$ for each setting of the hyper-parameters after applying the proposed vanilla AM method.

The experiments also feed mel-spectrogram excerpts extracted from the Jamendo training dataset to the machine listening models and record the activations of the output layer neurons for each excerpt, and select the top $N = 50$ excerpts with maximum activation for each output layer neuron. Thus, the experiments select 50 excerpts and their activations for SVDNet-R1 and 100 excerpts (50 for each of the output layer neurons) and their activations for SVDNet-R2. Additionally, for the SVDNet-R1 model, the experiment also records 50 excerpts and their activations that minimally activate the single output layer neuron. The experiments generate a new excerpt of 115 consecutive mel-spectrogram frames for every 50 time frames (around 0.7 seconds) in a recording. To perform GAN-based vanilla AM, the experiments optimise the objective in Equation 5.2 using the Adam optimiser with $\beta_1 = 0.99$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

5.4 Results

This section describes the results of the experiments that aim to analyse the effectiveness of the proposed techniques by applying them to the two deep SVD models. For each model, this section first describes results examining the effectiveness of the hyper-parameter selection method (see Section 5.2.4) and then uses the best hyper-parameter configuration, and the proposed vanilla AM method (see Section 5.2.2) to qualitatively analyse features that maximally (or minimally) activate the output layer neurons in both the models. Importantly, the qualitative analysis includes auralising the synthesised examples which in this section is done by the author. Here hyper-parameter configuration refers to the tuple of the three AM hyper-parameters: initial learning rate, regularisation parameter (GAN prior weight), and number of iterations. The results will include discussions about the interpretability of the examples synthesised by the

Experiment label	Configuration label	l_r	λ	N_t	FID
Maximise	C_1^{max}	0.01	0.001	100	1.654
	C_2^{max}	0.01	0.1	1000	33.087
	C_3^{max}	0.1	0.001	1000	1557.733
Minimise	C_1^{min}	0.001	0.1	1000	0.541
	C_2^{min}	0.001	0.1	100	16.086
	C_3^{min}	0.1	0.001	1000	60533.079

Table 5.3: The best, median, and worst hyper-parameter configurations according to the proposed FID-based evaluation metric, computed using activations of the inputs synthesised to maximally (or minimally) activate the output neuron in SVDNet-R1. For example, C_1^{max}, C_2^{max}, and C_3^{max} represent the best, median, and worst configurations for the experiment with label "maximise". l_r, λ, N_t, and FID indicate the initial learning rate, GAN prior weight, number of optimisation iterations, and the resulting value of the evaluation metric, respectively.

proposed methods. Thus, it is important to understand that in the context of this thesis an example is interpretable if one can recognise sound(s) or sound characteristics present in it.

5.4.1 Analysing the output neuron in SVDNet-R1

This section describes the results of the experiments to analyse the SVDNet-R1 model. Specifically, the experiments aim to understand the features that maximally and minimally activate the output layer neuron in the model. To do that, the experiments first identify the best hyper-parameter configurations for maximising and minimising the pre-sigmoidal activation of the output neuron and then use the selected configurations to synthesise mel-spectrograms that maximally (or minimally) activate the output neuron.

5.4.1.1 Hyper-parameter configuration selection

This section discusses the results of two experiments that use the FID-based evaluation metric to rate different hyper-parameter configurations for maximally and minimally activating the output neuron in SVDNet-R1. Additionally, the experiments qualitatively analyse whether the metric reflects the interpretability of the generated examples.

Experiment 1: Maximising the neuron activation

The first experiment iteratively optimises an i_{th} noise vector $\tilde{z}_i$ by using the GAN-based vanilla AM method to maximally activate the output neuron. Thus,

Experiment label	Seed	a_{inp}	a_1	a_2	a_3
Maximise	4	1.03	7.94	14.47	64.48
	10	2.81	9.26	15.43	77.96
	21	−0.86	5.51	12.87	20.19
	39	1.14	8.09	11.88	87.60
Minimise	7	−3.74	−8.34	−5.06	−565.74
	12	−0.49	−4.94	−2.47	−372.81
	30	1.81	−4.83	−2.49	−18.95
	49	1.53	−5.86	−3.19	−489.47

Table 5.4: The pre-sigmoidal activations of the output neuron in SVDNet-R1 for each mel-spectrogram in Fig. 5.3 and Fig. 5.4. a_{inp} refers to the activation value for the initial mel-spectrogram (first output from the GAN). a_1, a_2, and a_3 refer to the activation values for mel-spectrograms synthesised using the best, median, and worst hyper-parameter configurations, respectively. Seed refers to the number used to initialise the pseudorandom number generator.

for the set of 50 noise vectors for each hyper-parameter configuration (see Section 5.3.4), the experiment synthesises 50 mel-spectrograms (one for each noise vector) and records their pre-sigmoidal activations. The experiment repeats the above step for each of the 27 hyper-parameter configurations generating a distribution of 50 activations for each configuration. The experiment uses the same distribution of seed values for sampling initial noise vectors for each hyper-parameter configuration. Additionally, the experiment also records activations of the top 50 input mel-spectrogram excerpts from the Jamendo training dataset that maximally activate the output neuron. Finally, the experiment uses Equation 5.3 to compute the FID between the distribution of activations corresponding to the real data and the distribution of activations corresponding to each hyper-parameter configuration.

Table 5.3 mentions the best (C_1^{max}), the median (C_2^{max}), and the worst (C_3^{max}) hyper-parameter configurations and their FID scores out of the 27 hyper-parameter configurations the experiment analyses. The best hyper-parameter configuration is the one that generates examples with activation distribution closest (minimum FID) to the activation distribution from the real data. The experiment further investigates whether FID reflects the interpretability of the generated examples. In other words, the experiment analyses whether the examples synthesised using the best configuration are more interpretable than those synthesised using the worst configuration.

The experiment investigates the above question by visualising and auralising mel-spectrograms synthesised using each of three hyper-parameter configurations. Specifically, the experiment first samples four noise vectors $\tilde{z}_1, \tilde{z}_2, \tilde{z}_3$, and $\tilde{z}_4$, each using a randomly selected seed and then uses the best, median and worst hyper-parameter configurations corresponding to the experiment la-

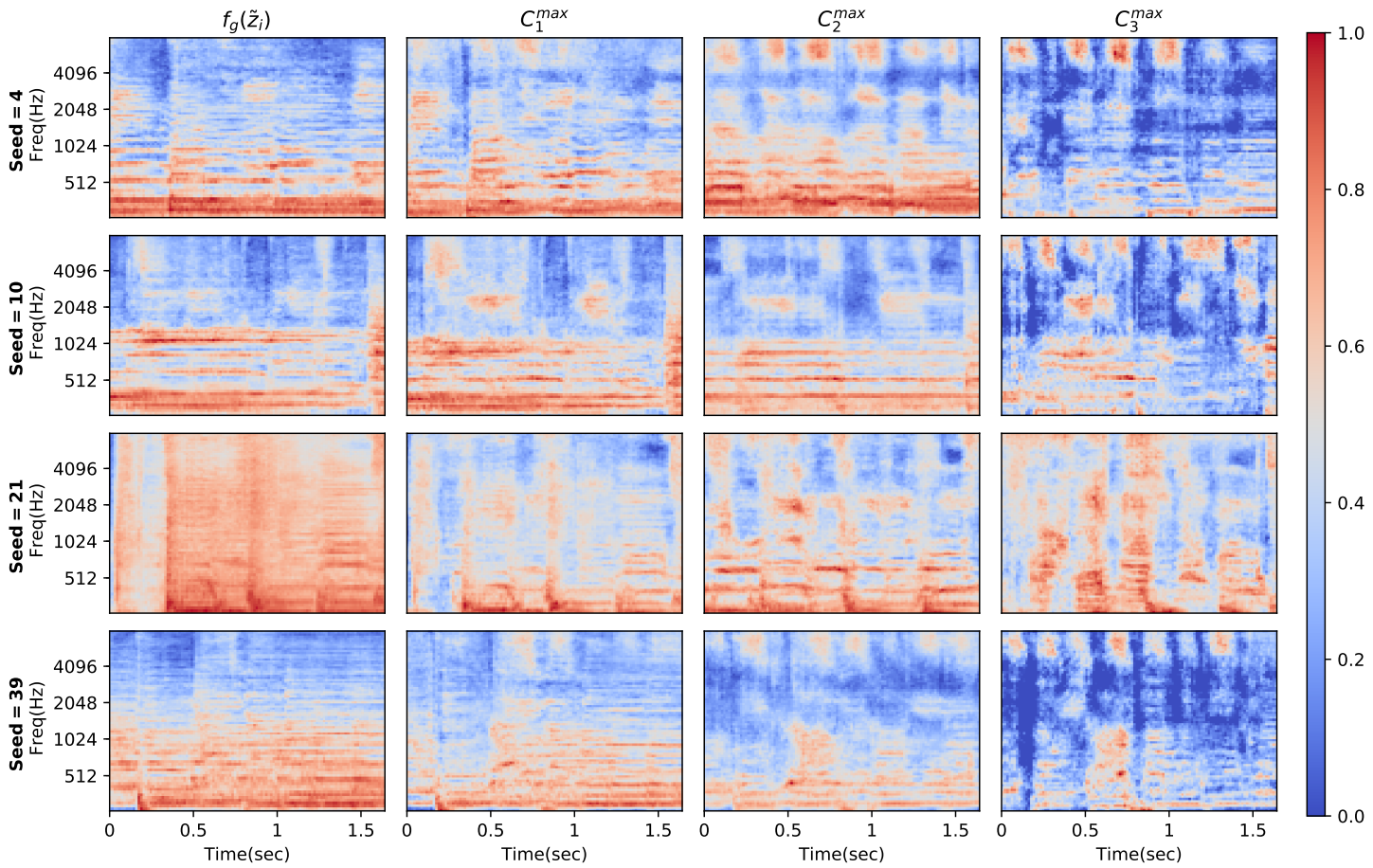

Figure 5.3: Mel-spectrogram visualisations demonstrating the effectiveness of the proposed evaluation metric. Each mel-spectrogram is normalised in scale independently so that red colours show relatively high and blue colours show relatively low spectral energy. The leftmost column shows the first output of the GAN $f_g(\tilde{z}_i)$ for four initial noise vectors $\tilde{z}_1, \tilde{z}_2, \tilde{z}_3$, and $\tilde{z}_4$ (one per row), each sampled using a different seed. The others show the result of applying the GAN-based vanilla AM method to maximally activate the output neuron in SVDNet-R1 using three different hyper-parameter configurations C_1^{max}, C_2^{max}, and C_3^{max} that represent the best, median, and worst configuration from the set of 27 configurations according to the evaluation metric, respectively. Seed refers to the number used to initialise the pseudorandom number generator.

bel "maximise" from Table 5.3 to generate mel-spectrograms. Figure 5.3 shows the results of the experiment. For each noise vector $\tilde{z}_i$ (a row in the figure), the figure depicts the first GAN output $f_g(\tilde{z}_i)$ and the synthesised mel-spectrograms after iteratively maximising the output neuron activation for N_t iterations using the C_1^{max}, C_2^{max}, and C_3^{max} configurations. Table 5.4, experiment label "maximise" reports the pre-sigmoidal activations from the output neuron for each mel-spectrogram in Fig. 5.3.

To auralise the generated examples, the experiment first maps each mel-spectrogram to the corresponding magnitude spectrogram using the Moore-Penrose pseudoinverse of the mel-filterbank matrix [Boucheron and De Leon, 2008] and then uses the Griffin-Lim [Griffin and Lim, 1984] and the inverse discrete Fourier transform (IDFT) [Müller, 2015] algorithms to synthesise the corresponding phase spectrogram and to map the resulting complex spectrogram to the temporal domain. Appendix B.2 mathematically explains the mel-spectrogram inversion procedure and discusses its lossy nature. Moreover, the appendix qualitatively demonstrates that although the inversion procedure involves information loss, the synthesised audio is interpretable.

The qualitative analysis of the generated mel-spectrograms in Fig. 5.3 provides insights into the behaviour of the proposed AM algorithm and the hyper-parameter selection method. The results demonstrate that expectedly the AM algorithm is sensitive to hyper-parameter values as for each seed the synthesised examples have different levels of interpretability and maximally activate the output neuron to very different activation values. For example, for nearly all the seeds, the best configuration C_1^{max} generates mel-spectrograms having more harmonic content and high-frequency energy than the initial GAN outputs. Moreover, all the examples are interpretable with the bin energy distribution range similar to that in mel-spectrograms used for training the GAN. The results for seed = 21 seem to slightly deviate from the above behaviour. However, it is important to note that $f_g(\tilde{z}_{21})$ is noisy and the example synthesised using C_1^{max} cleans up the noise during optimisation.

The median configuration C_2^{max} provides mixed results. For seeds 4 and 21, it generates interpretable audio with increased harmonic content and high-frequency energy. However, for the other seeds, the examples from C_2^{max} are sparser and less realistic than those from C_1^{max}. A plausible explanation for mixed results from C_2^{max} is that the vocal category in the real distribution contains both vocals and non-vocals. The presence of vocals increases the output neuron activation and the presence of non-vocals decreases it. Thus, the top N activations from the real data do not contain very high activations, which may happen if the vocal category also contains inputs with only vocals. Thus, depending on the content in $f_g(\tilde{z}_i)$, the median input will provide different types

of results, as for some inputs, there will be more scope for optimisation.

Finally, the results demonstrate that the AM algorithm generates adversarial examples using the worst configuration C_3^{max} [Nguyen et al., 2015]. Such examples activate the output neuron to very high values (e.g., the activation value for seed = 39 is 87.60, see Table 5.4), but are extremely sparse with maximum energies one order of magnitude greater than those in mel-spectrograms used for GAN training. The example corresponding to seed 21 is the least adversarial of all, however, its bin energy distribution range is unrealistic, and its temporal representation is uninterpretable.

Experiment 2: Minimising the neuron activation

To further validate the effectiveness of the proposed hyper-parameter selection method, this section repeats the previous experiment, but instead of maximising, it minimises the activation of the output neuron. Thus, for each of the 27 hyper-parameter configurations, experiment two synthesises 50 mel-spectrograms by iteratively minimising the pre-sigmoidal activation of the output neuron. Moreover, the experiment selects the top 50 minimally activating excerpts from the training dataset. Finally, the experiment computes the FID between the activation distribution from the real dataset and the activation distribution corresponding to each hyper-parameter configuration. Table 5.3, experiment label "minimise" reports the best (C_1^{min}), median (C_2^{min}), and worst (C_3^{min}) configurations. The best configuration is the one with the smallest FID, suggesting that the activation distribution for this configuration is very close to the activation distribution from the real data, assisting in the generation of realistic examples.

Experiment two further analyses the relation between the FID-based metric and example interpretability by visualising and auralising the synthesised mel-spectrograms. To do that, the experiment uses the C_1^{min}, C_2^{min}, and C_3^{min} configurations to iteratively optimise four different noise vectors to synthesise mel-spectrograms (three for each noise vector) that minimally activate the output neuron. Fig. 5.4 depicts the results of the experiment. The visualisations for each noise vector show the initial GAN output, followed by examples synthesised using the best, median and worst configurations, respectively. Table 5.4, experiment label "minimise" reports the pre-sigmoidal activations for each mel-spectrogram in Fig. 5.4.

The results demonstrate that for all the four cases, the worst configuration generates adversarial (unrealistic and uninterpretable) examples that minimise the neuron activation to very low values (e.g., for seed 7, the activation is -565.74), but are extremely sparse with the bin energy distribution two orders of magnitudes higher than for the mel-spectrograms used for GAN training.

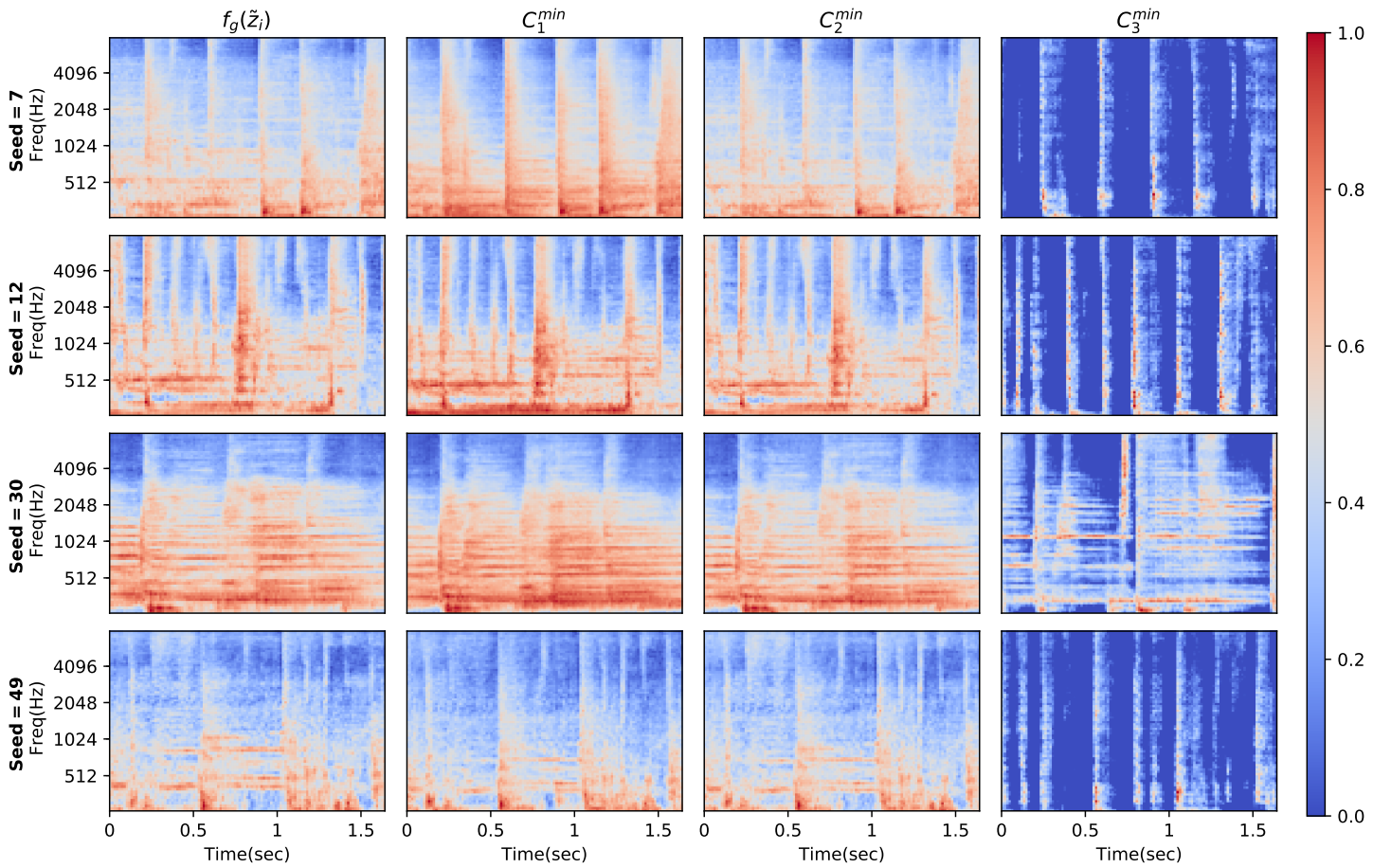

Figure 5.4: Mel-spectrogram visualisations demonstrating the effectiveness of the proposed evaluation metric. Each mel-spectrogram is normalised in scale independently so that red colours show relatively high and blue colours show relatively low spectral energy. The leftmost column shows the first output of the GAN $f_g(\tilde{z}_i)$ for four noise vectors $\tilde{z}_1, \tilde{z}_2, \tilde{z}_3$, and $\tilde{z}_4$ (one per row), each sampled using a different seed. The others show the result of applying the GAN-based vanilla AM method to minimally activate the output layer neuron in SVDNet-R1 using three different hyper-parameter configurations C_1^{min}, C_2^{min}, and C_3^{min} that represent the best, median, and worst configuration from the set of 27 configurations according to the evaluation metric, respectively. Seed refers to the number used to initialise the pseudorandom number generator.

On the other hand, in nearly all the cases, the best configuration generates realistic examples that minimise the neuron activation to a low value. These examples contain increased harmonic content and more prominent onsets than their corresponding first GAN outputs. The median configuration synthesises interpretable examples that seem quite similar to their corresponding first GAN outputs. However, all the examples have pre-sigmoidal activations more than those from the best configuration (see Table 5.4), suggesting C_2^{min} is less suitable than C_1^{min}.

The results from the two experiments demonstrate that the proposed hyper-parameter selection method effectively identifies the best and worst hyper-parameter configurations for AM. Moreover, the results also suggest that in addition to the best configuration, the proposed method also assists in identifying other suitable configurations that may synthesise interpretable examples for some noise vectors. Although the experiments demonstrated the proposed hyper-parameter selection method for GAN-based vanilla AM, the proposed approach is generic and thus applicable to other vanilla AM approaches.

5.4.1.2 Qualitative analysis of explanations

This section discusses the experiment that aims to understand features that the output layer neuron in the SVDNet-R1 model captures. To do that, the experiment uses the best configurations from the previous activation maximisation and minimisation experiments (C_1^{max} and C_1^{min} in Table 5.3), to produce positive and negative examples for the output layer neuron by maximising and minimising its pre-sigmoidal activations, respectively. Since the SVDNet-R1 model aims to distinguish vocals from non-vocals, the experiment in this section investigates whether the output layer neuron captures high-level class concepts. In other words, the experiment analyses whether the positive and negative examples synthesised using the proposed GAN-based vanilla AM method for the output layer neuron indicate the presence and absence of vocals, respectively.

Figure 5.5 shows the results of the experiment. Each column in the figure corresponds to a noise vector $\tilde{z}_i$, sampled using a random seed. Each column shows the initial GAN output (row label $f_g(\tilde{z}_i)$) and three mel-spectrograms synthesised using the same noise vector as initialisation point. The examples corresponding to the row labels "Maximise[1]" and "Minimise", maximally and minimally activate the output neuron using the configurations C_1^{max} and C_1^{min}, respectively. The experiment also generates examples to maximally activate the output neuron using the median configuration C_2^{max} (row label "Maximise[2]") as the previous experiment suggested that in some cases C_2^{max} generates more interpretable examples than C_1^{max} (see Section 5.4.1.1).

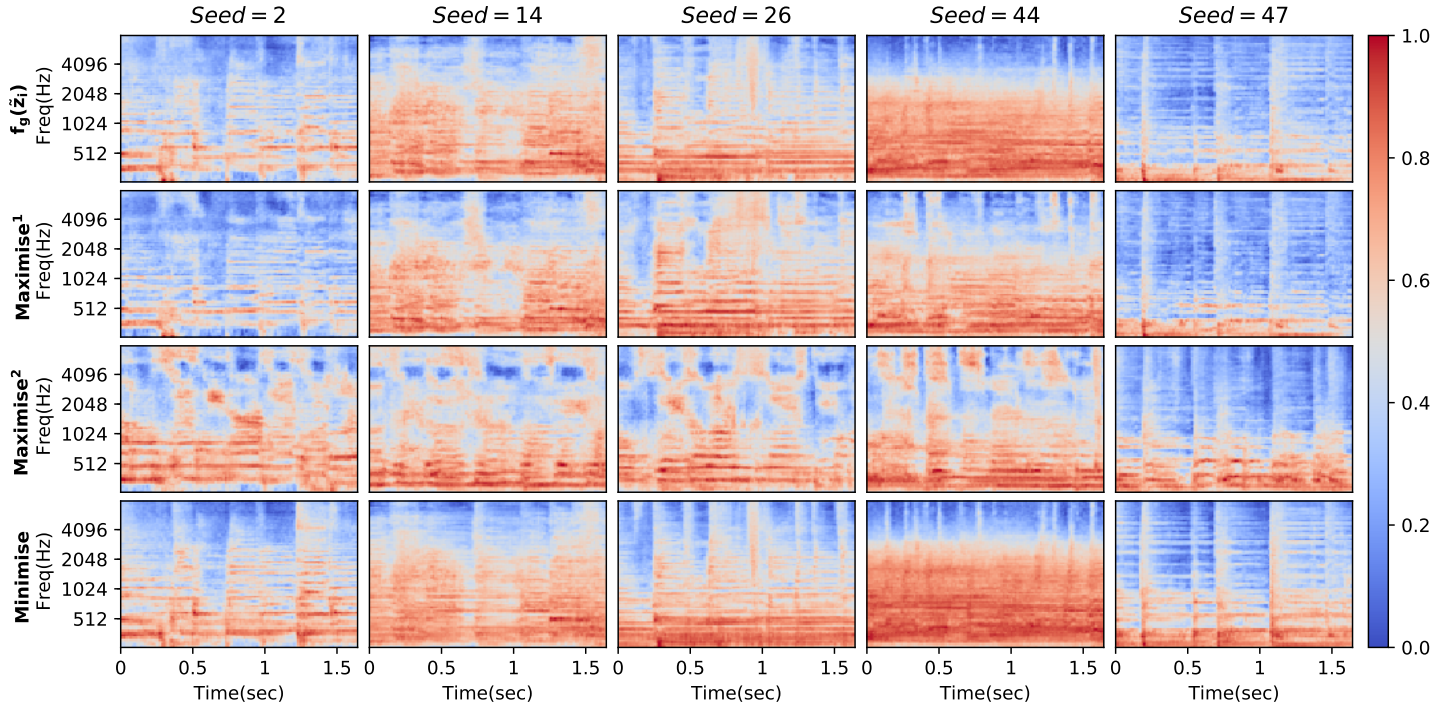

Figure 5.5: Visualisations depicting mel-spectrograms that maximally or minimally activate the output neuron in the SVDNet-R1 model. Each mel-spectrogram is normalised in scale independently so that red colours show relatively high and blue colours show relatively low spectral energy. The top row represents initial GAN outputs $f_g(\tilde{z}_i)$ for five initial noise vectors $\tilde{z}_1, \tilde{z}_2, \tilde{z}_3, \tilde{z}_4$, and $\tilde{z}_5$, each sampled with a different seed. The second and third rows represent examples synthesised by maximally activating the output neuron using the configurations C_1^{max} and C_2^{max} from Table 5.3. The last row depicts mel-spectrograms synthesised by minimally activating the output neuron using the configuration C_1^{min} from Table 5.3. Seed refers to the number used to initialise the pseudorandom number generator.

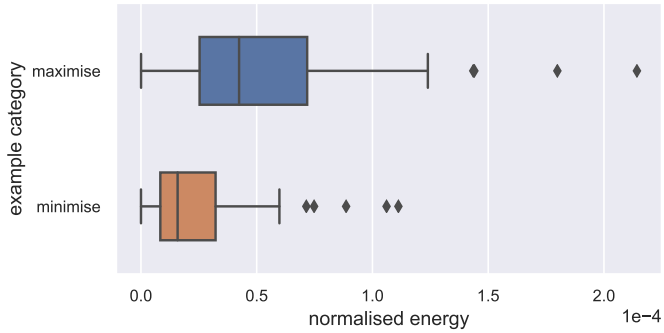

Figure 5.6: The figure depicts the distribution of normalised energy for frequencies > 4000 Hz for each set of 50 examples that maximally and minimally activate the single output neuron in SVDNet-R1, respectively.

The visualisations for most of the positive examples depict a stronger presence of harmonic content, a lack of energy in the very low frequency band below the human voice range, and few transient sounds such as drum hits (visible as vertical bars), indicating that the positive explanations indeed have many characteristics typical of vocal content. In contrast, the negative examples have stronger transients and more bass frequency content, indicating the successful generation of instrumental examples. Moreover, all the positive examples except those corresponding to seed $= 47$, depict the presence of energy in higher frequencies (frequency ≥ 4096), and all the negative examples depict the absence of energy in those frequencies. This also seems a vocal characteristic as singing voice may contain fricatives that result in energy in higher frequencies. This observation is further analysed using a dataset of 100 examples, where 50 examples are synthesised by maximally and minimally activating the output neuron, respectively. The experiment uses seeds $\in \{0, ..., 49\}$ and generates two examples for each seed, one by maximising and the other by minimising the activation of the output neuron. For each example, the normalised energy in frequencies > 4000 Hz is computed, where normalisation is done using the global energy level of the example. Fig. 5.6 presents a plot depicting the distribution of normalised energy for a set of 50 examples corresponding to each example category. The plot further supports the earlier observation that maximally activating examples have comparatively more energy in frequencies > 4000 Hz than the minimally activating examples.

Interestingly, Chapter 6 discusses very similar results from another experiment that uses feature inversion to analyse input content that the deepest hidden layer preserves in the SVDNet-R1 model. In addition to visualising the synthesised mel-spectrograms, the experiment also auralises temporal signals obtained

mel-spectrogram category	a_1	a_2	a_3	a_4	a_5
$f_g(\tilde{z}_i)$	1.97	2.40	0.00	−0.28	−0.49
Maximise[1]	8.21	6.88	7.92	6.32	4.29
Maximise[2]	17.3	15.99	17.04	12.52	6.53
Minimise	−5.08	−5.24	−5.94	−4.94	−7.2

Table 5.5: The pre-sigmoidal activations of the output layer neuron in SVDNet-R1 for each mel-spectrogram in Fig. 5.5. a_1, a_2, a_3, a_4, and a_5 refer to activation values corresponding to mel-spectrograms synthesised using noise vectors sampled with seeds $2, 14, 26, 44$, and 47, respectively.

by inverting the mel-spectrograms using the steps mentioned in Appendix B.2. The listening tests also confirm the presence and absence of vocals in the positive and negative examples, respectively. It is important to note that these listening tests are informal listening tests carried out by the author. Section 5.5 presents a perceptual study that further analyses these observations.

The above results demonstrate that the proposed GAN-based vanilla AM approach can generate effective explanations that provide useful insights into the concepts acquired by deep machine listening models. Moreover, analysis of explanations for the output neuron confirms that similar to image classification models, SVDNet-R1 learns high-level class concepts in that neuron. Furthermore, Table 5.5 demonstrates that the proposed AM method effectively optimises the pre-sigmoidal activation of the output layer neuron in all the cases.

5.4.2 Analysing the output neurons in SVDNet-R2

The previous experiment demonstrated that the output neuron in the SVDNet-R1 model captures high-level class concepts. However, as the single neuron needs to learn about both the classes, the class representations it learns can be fairly complex, involving the overlap of different inter-class and intra-class concepts. Such characteristics may affect the interpretability of examples synthesised using the AM algorithm. On the other hand, if a neuron learns only about a single class, the examples AM synthesises for that neuron may be more interpretable than those for the single neuron model. This section focuses on qualitative interpretability by visualising and auralising the synthesised examples. It analyses this hypothesis by using the proposed AM method (see Section 5.2.2) to understand the concepts the output neurons capture in the SVDNet-R2 model. This model differs from the SVDNet-R1 model in only the number of neurons in its output layer (see Chapter 3). SVDNet-R2 contains two neurons with indices 0 (non-vocal) and 1 (vocal) in its output layer. Thus, the experiment to analyse SVDNet-R2 uses the proposed AM method to synthesise mel-spectrograms that maximally activate the pre-softmax activations of each neuron indepen-

Neuron index	Neuron label	Configuration label	l_r	λ	N_t	FID
0	Non-vocal	C_1^{N0}	0.001	0.1	1000	3.894
		C_2^{N0}	0.01	0.1	1000	28.596
		C_3^{N0}	0.1	0.001	1000	88626.081
1	Vocal	C_1^{N1}	0.01	0.001	100	8.459
		C_2^{N1}	0.01	0.1	1000	233.024
		C_3^{N1}	0.1	0.001	1000	26751.636

Table 5.6: The best (C_1^{Nj}), median (C_2^{Nj}), and worst (C_3^{Nj}) hyper-parameter configurations for maximally activating the neuron with index j in the output layer of the SVDNet-R2 model. The configurations are selected using the proposed FID-based evaluation metric. l_r, λ, N_t, and FID indicate the learning rate, GAN prior weight, number of optimisation iterations, and the resulting value of the evaluation metric, respectively.

dently. The experiment first uses the hyper-parameter selection method from section 5.2.4 to select the best hyper-parameter configuration to perform AM for each neuron and then uses the selected configurations to synthesise examples.

5.4.2.1 Hyper-parameter configuration selection

This section describes the experiments to select the best hyper-parameter configuration to perform AM for each output layer neuron in SVDNet-R2. To do that, for each output neuron, the experiment follows the procedure mentioned in Section 5.3.4. The experiment generates 50 examples for each of the 27 hyper-parameter configurations. The experiment generates each example by iteratively optimising a noise vector to maximise the pre-softmax activation of the target neuron. The experiment uses the same distribution of seeds to sample 50 noise vectors for each configuration. Additionally, the experiment selects the top 50 mel-spectrogram excerpts from the Jamendo training dataset that maximally activate the target neuron. Finally, for each neuron, the experiment computes the FID corresponding to each configuration by using activations from the real and synthesised examples (for that configuration).

Table 5.6 reports the results of the above experiment for both the output neurons. For each neuron, the table lists the best, median, and worst configurations using the FID-based metric, where the best configuration is the one with the smallest FID. Interestingly, the best, median, and worst configurations for the vocal neuron are same as those for maximising the neuron activation in SVDNet-R1, but with different (comparatively much larger) FID scores (see Table 5.3). Similarly, the best and worst configurations for the non-vocal neuron are the same as those for minimising the output neuron activation in SVDNet-R1, but again with different FID scores.

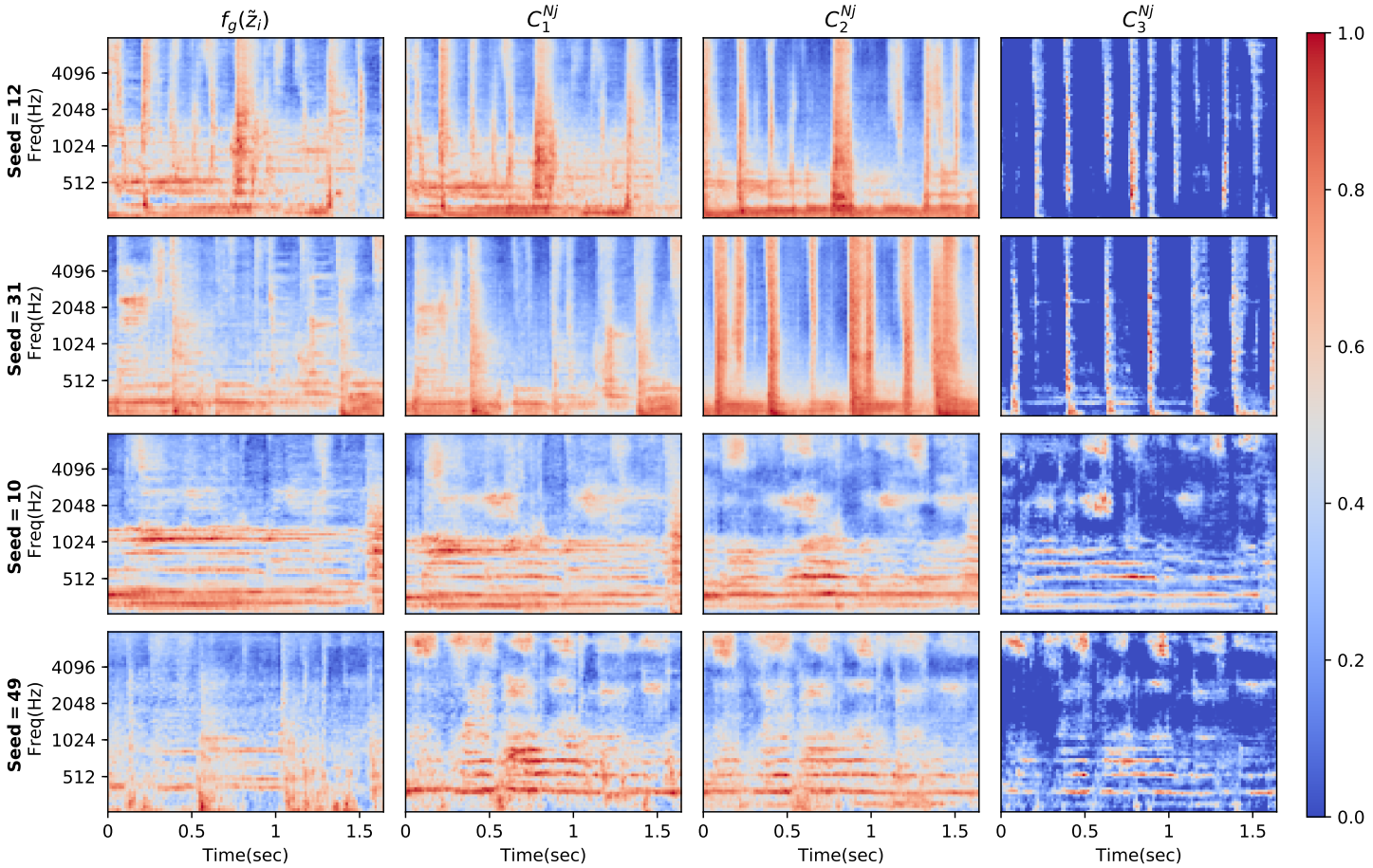

Figure 5.7: Mel-spectrogram visualisations demonstrating the effectiveness of the proposed evaluation metric. Each mel-spectrogram is normalised in scale independently so that red colours show relatively high and blue colours show relatively low spectral energy. The leftmost column shows the first output of the GAN $f_g(\tilde{z}_i)$ for four initial noise vectors $\tilde{z}_1, \tilde{z}_2, \tilde{z}_3$, and $\tilde{z}_4$ (one per row), each sampled using a different seed. The others show the result of applying the GAN-based vanilla AM method to maximally activate the non-vocal (index = 0) and vocal (index = 1) neurons in the output layer of SVDNet-R2 using three different hyper-parameter configurations C_1^{Nj}, C_2^{Nj}, and C_3^{Nj} that represent the best, median, and worst configuration from the set of 27 configurations according to the evaluation metric, respectively, where j indicates the neuron index. The two top and bottom rows depict the results for the non-vocal and vocal neurons, respectively. Seed refers to the number used to initialise the pseudorandom number generator.

Neuron index	Neuron label	Seed	a_{inp}	a_1	a_2	a_3
0	Non-vocal	12	1.27	5.83	10.7	384.17
		31	−1.53	5.9	16.91	497.06
1	Vocal	10	3.54	14.67	26.1	200.10
		49	0.33	24.85	37.5	373.82

Table 5.7: The pre-softmax activations of the non-vocal and vocal neurons in the output layer of SVDNet-R2 for the corresponding mel-spectrograms in Fig. 5.7. a_{inp} refers to the activation value for the initial mel-spectrogram (first output from the GAN). a_1, a_2, and a_3 refer to the activation values for the mel-spectrograms synthesised using the hyper-parameter configurations C_1^{Nj}, C_2^{Nj}, and C_3^{Nj}, respectively, where j indicates the neuron index. Seed refers to the number used to initialise the pseudorandom number generator.

The experiment also test whether the conclusion from the analysis of the SVDNet-R1 model about the relation between the FID-based metric and example interpretability extends to another machine listening model. To do that, for each neuron, the experiment first samples two noise vectors using two random seeds and then uses them and the proposed AM method to generate two examples that maximally activate the target neuron for each of the three configurations. Fig. 5.7 depicts the results of the experiment. The first two rows depict the results for the non-vocal neuron and the rest for the vocal neuron. Each row provides visualisations for the first GAN output and the examples corresponding to the best, median and worst configurations for the target neuron. Table 5.7 reports the pre-softmax activations from the corresponding neuron in the SVDNet-R2 model for each mel-spectrogram in Fig. 5.7. Additionally, the experiment also auralises all the temporal signals obtained from inverting all the mel-spectrograms in Fig. 5.7.

The results demonstrate that the worst configuration for each neuron synthesises adversarial examples that activate neurons to very high values, but are highly sparse, have unnatural bin energy distributions, and are perceptually uninterpretable. On the other hand, the best configuration for each neuron synthesises realistic and perceptually interpretable examples. Finally, the median configuration for each neuron provides mixed results. For example, for the non-vocal neuron, the median configuration generates interpretable and uninterpretable examples for the seeds 12 and 31, respectively. These results suggest that the proposed hyper-parameter selection method generalises to SVDNet-R2 and the FID-based metric relates to the interpretability of synthesised examples.

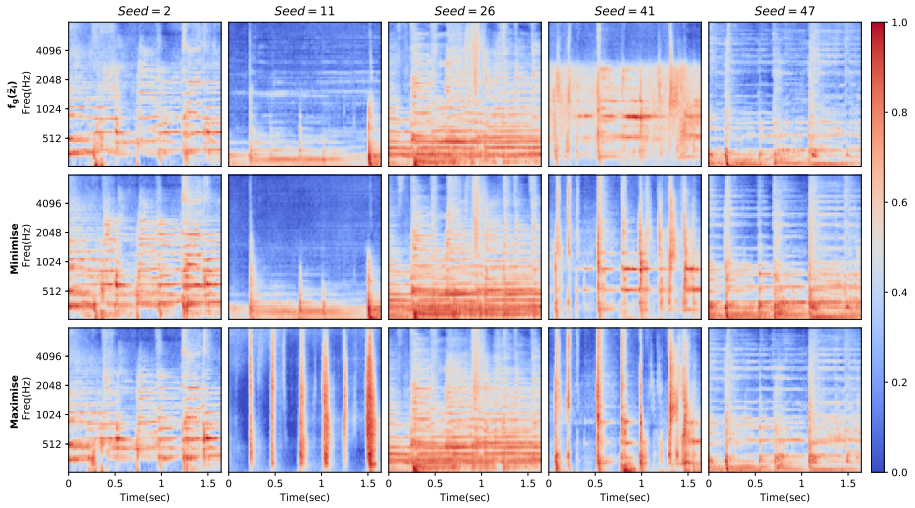

Figure 5.8: Mel-spectrogram visualisations illustrating the non-vocal concepts the output layer neurons learn in two deep SVD models. Each mel-spectrogram is normalised in scale independently so that red colours show relatively high and blue colours show relatively low spectral energy. The top row represents initial GAN outputs $f_g(\tilde{z}_i)$ for five noise vectors $\tilde{z}_1, \tilde{z}_2, \tilde{z}_3, \tilde{z}_4$, and $\tilde{z}_5$, each sampled using a different seed. The middle row depicts examples synthesised by minimally activating the output neuron in SVDNet-R1 using C_1^{min} from Table 5.3. The last row depicts mel-spectrograms synthesised by maximally activating the non-vocal neuron (index = 0) in SVDNet-R2 using C_1^{N0} from Table 5.6. Seed refers to the number used to initialise the pseudorandom number generator.

5.4.2.2 Qualitative analysis of examples

This section uses the best hyper-parameter configurations from the previous experiment to synthesise examples that maximally activate the output layer neurons in the SVDNet-R2 model. The experiment qualitatively analyses the synthesised examples to validate if the neurons learn high-level class concepts. Additionally, the experiment qualitatively compares the interpretability of examples synthesised for each neuron in the SVDNet-R2 model with corresponding examples for the SVDNet-R1 model.

To synthesise examples for the SVDNet-R2 model, the experiment samples five noise vectors for each output neuron by using five randomly selected seeds (a number between 1 and 50). The experiment uses a different distribution of seeds for each neuron to expand the qualitative analysis to more examples. The experiment uses the proposed AM method and the best configurations: C_1^{N0} for the non-vocal neuron and C_1^{N1} for the vocal neuron (see Table 5.6), to iteratively optimise the noise vectors and generate mel-spectrograms that maximally activate each output neuron independently.

Model	Mel-spectrogram category	a_1	a_2	a_3	a_4	a_5
SVDNet-R1	$f_g(\tilde{z}_i)$	1.97	−3.47	0.00	−0.36	−0.49
	Minimise	−5.08	−6.1	−5.94	−7.98	−7.2
SVDNet-R2	$f_g(\tilde{z}_i)$	−0.78	2.41	−0.93	−0.17	1.38
	Maximise	6.6	16.65	4.85	10.22	7.34

Table 5.8: The pre-sigmoidal and pre-softmax activations of the output layer neuron in SVDNet-R1 and the output layer non-vocal neuron (index= 0) in SVDNet-R2, respectively for each mel-spectrogram in Fig. 5.8. a_1, a_2, a_3, a_4, and a_5 refer to activation values corresponding to mel-spectrograms synthesised using noise vectors sampled with seeds $2, 11, 26, 41$, and 47, respectively. $f_g(\tilde{z}_i)$ refers to the first output from the GAN for an initial noise vector $\tilde{z}_i$.

Fig. 5.8 and Fig. 5.10 depict the results for the non-vocal and vocal neurons in the SVDNet-R2 model, respectively. In both the figures, each column depicts the initial GAN output and the examples corresponding to the SVDNet-R1 and SVDNet-R2 models, respectively. Additionally, for each mel-spectrogram in both the figures, Table 5.8 and Table 5.9 mention the activations of the non-vocal and vocal neurons in the SVDNet-R2 model, respectively. Furthermore, the experiment auralises the mel-spectrograms by inverting them to the temporal domain using the inversion steps mentioned in Appendix B.2.

The qualitative analysis of results suggests that the non-vocal and vocal neurons learn high-level class concepts. For example, all mel-spectrograms corresponding to the non-vocal neuron depict the presence of very prominent instrumental onsets. On the other hand, all mel-spectrograms corresponding to the vocal neuron depict the presence of vocal characteristics: high harmonic content, presence of energy in high-frequency regions, and reduced presence of instrumental onsets. The results also suggests that the proposed AM method seems to be useful for the analysis of other machine listening models as the examples it synthesises for analysing the SVDNet-R2 model are interpretable.

Similar to Section 5.4.1.2, this section presents a further investigation of the above observation that examples corresponding to the two neurons in SVDNet-R2 differ in the amount of energy in higher frequencies. To do that, first, 50 examples are synthesised using seeds $\in \{0, ..., 49\}$ and the best configurations for each neuron. Importantly, for each seed, the experiment generates two examples, one for each output neuron. Later, for each example, normalised energy is computed for frequencies $>$ 4000 Hz. Fig. 5.9 presents a plot depicting the distributions of normalised energy for the sets of examples corresponding to each neuron. The results further support the earlier observation that examples corresponding to the vocal and non-vocal neurons differ in the amount of energy in frequencies $>$ 4000 Hz, with vocal examples having more energy in those frequencies.

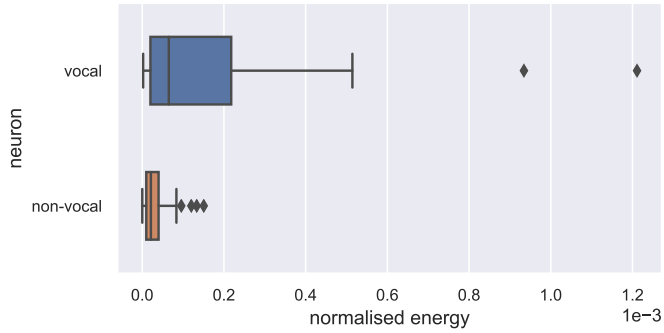

Figure 5.9: The figure depicts the distribution of normalised energy for frequencies > 4000 Hz for two sets of 50 examples, each corresponding to one of the two neurons in the SVDNet-R2 model.

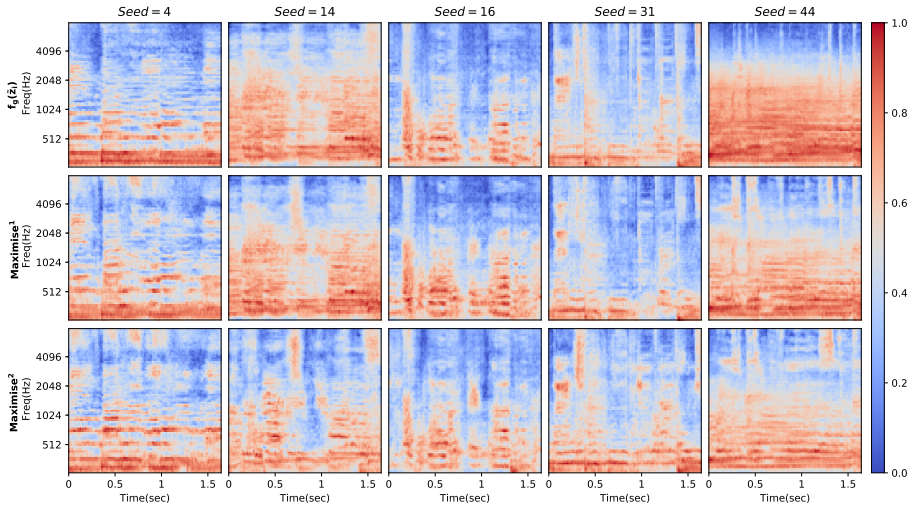

Figure 5.10: Mel-spectrogram visualisations illustrating the vocal concepts the output layer neurons learn in two deep SVD models. Each mel-spectrogram is normalised in scale independently so that red colours show relatively high and blue colours show relatively low spectral energy. The top row represents initial GAN outputs $f_g(\tilde{z}_i)$ for five noise vectors $\tilde{z}_1, \tilde{z}_2, \tilde{z}_3, \tilde{z}_4$, and $\tilde{z}_5$, each sampled using a different seed. The middle row depicts examples synthesised by maximally activating the output layer neuron in SVDNet-R1 using C_1^{max} from Table 5.3. The last row depicts mel-spectrograms synthesised by maximally activating the vocal output neuron (index $= 1$) in SVDNet-R2 using C_1^{N1} from Table 5.6. Seed refers to the number used to initialise the pseudorandom number generator.

The experiment also (qualitatively) compares the five mel-spectrograms for the non-vocal neuron in SVDNet-R2 (see Fig. 5.8) with corresponding mel-spectrograms that minimally activate the output neuron activation in SVDNet-

Model	Mel-spectrogram category	a_1	a_2	a_3	a_4	a_5
SVDNet-R1	$f_g(\tilde{z}_i)$	1.03	2.40	2.63	2.03	−0.28
	Maximise	7.94	6.88	8.27	9.07	6.32
SVDNet-R2	$f_g(\tilde{z}_i)$	0.98	1.14	2.30	1.30	−0.04
	Maximise	18.14	17.78	12.92	12.13	11.99

Table 5.9: The pre-sigmoidal and pre-softmax activations of the output layer neuron in SVDNet-R1 and the output layer vocal neuron (index= 1) in SVDNet-R2, respectively for each mel-spectrogram in Fig. 5.10. a_1, a_2, a_3, a_4, and a_5 refer to activation values corresponding to mel-spectrograms synthesised using noise vectors sampled with seeds $4, 14, 16, 31$, and 44, respectively. $f_g(\tilde{z}_i)$ refers to the first output from the GAN for an initial noise vector $\tilde{z}_i$.

R1. Similarly, the experiment compares the five mel-spectrograms for the vocal neuron in SVDNet-R2 (see Fig. 5.10) with corresponding mel-spectrograms that maximally activate the output neuron activation in SVDNet-R1. To synthesise examples for the SVDNet-R1 model, the experiment optimises the noise vectors used for the SVDNet-R2 model by using the best maximisation C_1^{max} and minimisation C_1^{min} configurations. The comparison of examples from the two models assists in understanding if using one neuron to learn high-level concepts corresponding to each class assists in synthesising more interpretable examples. The experiment hypothesises that using one neuron per class to learn high-level concepts corresponding to the class may assist the neuron in learning interpretable representations as it needs to focus only on intra-class concepts. The examples AM synthesises for an output neuron, to gain an insight into how the neuron represents a class, aims to capture such representations. This suggests that learning of interpretable class representations by an output neuron may assist in synthesising interpretable examples that may provide more insights into the invariances and intra-class concepts the neuron learns from data.

Fig. 5.8 and Fig. 5.10 provide visualisations depicting the synthesised mel-spectrograms for each deep SVD model. Table 5.8 and Table 5.9 mention the activations for each mel-spectrogram in the two figures. The results of the comparison between examples for the non-vocal neuron in SVDNet-R2 and examples that minimise the output neuron activation in SVDNet-R1 suggest that for seeds $2, 26, 41$ and 47 the examples from both the models have nearly the same characteristics, which seems due to the use of the same hyper-parameter configurations and noise vectors. However, for some cases, the examples from SVDNet-R2 seem a more refined version of examples from SVDNet-R1. For instance, the examples corresponding to seeds $2, 26$, and 47 visually differ in the amount of energy in lower frequencies. Specifically, the normalised energies (for frequencies < 250 Hz) for the three examples are $0.005, 0.002, 0.006$ and $0.006, 0.004, 0.007$ for the SVDNet-R2 and SVDNet-R1 models, respectively.

However, auralisation of those examples does not highlight major differences. In some cases, it suggests that the examples from SVDNet-R2 are slightly less noisy (more interpretable) than examples from SVDNet-R1, but overall, all the examples have clear instrumental onsets and (pair-wise) perceptually sound very similar. Interestingly, for seed 11, the example from the SVDNet-R2 model is comparatively less interpretable to that from the SVDNet-R1 model. The auralisation of the inverted mel-spectrogram corresponding to seed 11 and the SVDNet-R1 model suggests the presence of multiple instrumental (drum) onsets, but the inverted mel-spectrogram corresponding to the SVDNet-R2 model is harder to interpret. Thus, the previous results suggest that although some examples from SVDNet-R2 are slightly less noisy than the corresponding examples from SVDNet-R1, overall the output neuron in SVDNet-R1 can learn class representations that are similar to those learnt by the non-vocal neuron in SVDNet-R2.

The experiment further compares the examples synthesised for the vocal neuron in SVDNet-R2 and examples synthesised to maximally activate the output neuron in SVDNet-R1. The examples synthesised for both the models have (pair-wise) similar characteristics which again seems to be the influence of using the same hyper-parameter configurations and noise vectors. However, the results also highlight some differences in explanations from the two models. For example, for seeds $4, 14$ and 44, the examples from the SVDNet-R2 model are less noisy versions (have more interpretable vocal content) of the examples from the SVDNet-R1 model. Moreover, for these seeds, the change in activation value (final example activation - initial GAN activation) for the SVDNet-R2 model is at least two times higher than for the SVDNet-R1 model suggesting that SVDNet-R2 can optimise the noise vectors for these seeds much more than SVDNet-R1. However, overall the differences between the examples from the two models are not very prominent, suggesting that the output neuron in the SVDNet-R1 model can learn class representations that are fairly similar to those learnt by the vocal neuron in the SVDNet-R2 model. Thus, the results from the example comparison experiment suggest that despite the challenges of learning inter-class and intra-class concepts, the examples synthesised using AM for maximising and minimising the single neuron in SVDNet-R1 model are quite similar in interpretability to those for the SVDNet-R2 model.

5.5 Perceptual study

This section presents a perceptual study that aims to analyse two observations from the qualitative analysis experiments in Sections 5.4.1.2 and 5.4.2.2. The section first introduces the goal of the study (Section 5.5.1), then, it describes

the design of the study (Section 5.5.2), and finally, it presents the results of the study and highlights the key conclusions (Section 5.5.3). All participant data is kept private and secured safely and the study is approved by the Queen Mary Ethics of Research Committee (Ref: QMERC2447).

5.5.1 Goal

The perceptual study aims to investigate the three questions mentioned below

- **Q1:** Is the audio synthesised for maximally activating the SVDNet-R1 output neuron less intelligible than the audio synthesised for maximally activating the vocal output neuron in SVDNet-R2? An audio example is intelligible if one can recognise sound(s)/sound characteristics present in it. Additionally, this question is investigated for the non-vocal output neuron in SVDNet-R2. It is important to note that the study used the term "intelligible" instead of "interpretable" although with the same definitions as the former is an established term in the literature, referring to comprehending audio signals (e.g., speech) [Rennies and Schepker, 2014].
- **Q2:** Is the output neuron in SVDNet-R1 maximally and minimally activated by audio containing sound characteristics representative of singing voice and instrumental music, respectively?
- **Q3:** Do the proposed AM and hyper-parameter selection methods synthesise "intelligible" audio? Importantly, in this work, the recognition of sound(s) or their characteristics is agnostic to whether they are natural or synthetic. For example, a listener should recognise a sound as singing voice irrespective of whether it sounds natural or synthetic.

5.5.2 Perceptual study design

The perceptual study design involves three key components - a participant questionnaire, two listening tests, and audio stimuli.

5.5.2.1 Participant questionnaire

Before starting the listening tests, each participant read an information sheet and provided their consent for participation. Additionally, all participants filled a questionnaire consisting of six questions that collected information regarding their age, gender, whether they have any form of hearing impairment, are they using headphones for the tests, do they have a musical background, and do they have a machine learning background. The last four questions are yes or no questions and for the last two questions, the participants were presented

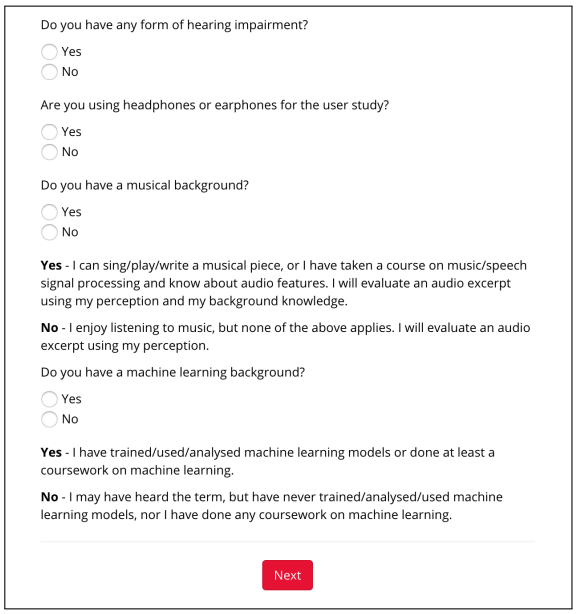

Figure 5.11: The figure depicts the last four yes/no questions in the participant questionnaire.

with definitions for each choice. Fig. 5.11 presents the last four questions in the questionnaire.

5.5.2.2 Listening tests

The study was conducted online using Gorilla experiment builder[3]. To participate in the study, all participants needed to have access to a computer with Chrome or Firefox browsers, a decent bandwidth internet and were strongly recommended to use an earphone/headphone in a quiet environment. The study involved two listening tests, each around $5-10$ mins and the participants were encouraged to participate in both the tests. Each test involved listening to several audio stimuli (examples) and answering the corresponding questions. Each audio stimulus was around 1.6 secs and the participants were allowed to listen to them as many times as they wanted.

Listening test 1 (LT_1): This test aimed to answer the first question in Section 5.5.1. The test presented each participant with ten pairs of audio examples, each pair containing examples synthesised by performing AM, starting from the same random seed, with each of the two SVD models. The first five and the last five pairs of excerpts correspond to the vocal and non-vocal neurons, respectively. Additionally, the test included three pairs of anchors, each

[3]https://gorilla.sc

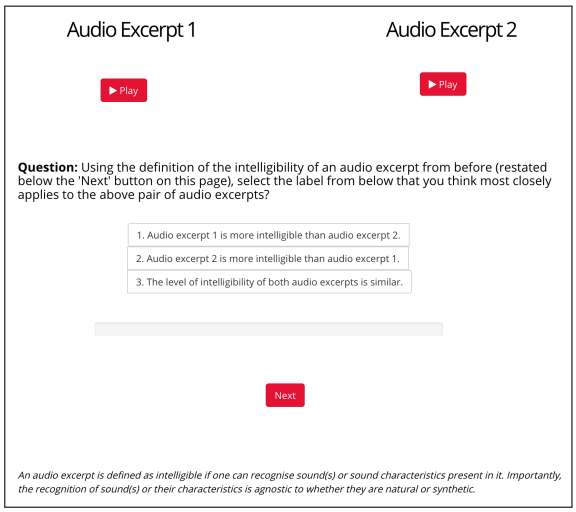

Figure 5.12: The figure depicts the user interface for listening test 1. Each audio excerpt corresponds to one of the two SVD models.

containing the same white noise, real vocals, and real non-vocals in both the excerpts, respectively. Anchors were added to help identify and ignore responses from participants providing random responses. Thus, in LT_1, each participant listened to 26 (13 pairs of) audio examples, 20 synthetic from AM and 6 real. The test presented example pairs randomly to the participants. For each pair, a participant first listened to the two examples and then answered a question comparing their intelligibility. Fig. 5.12 depicts the user interface for LT_1

Listening test 2 (LT_2): This test aimed to answer the last two questions in Section 5.5.1. LT_2 required each participant to listen to thirteen audio examples, ten synthesised using the AM method and three anchors. As in LT_1, the three anchors are white noise, real vocal, and real non-vocal examples, respectively. In the ten synthetic examples, five maximally activated and the remaining five minimally activated the output neuron in SVDNet-R1. The examples were presented one by one, randomly to each participant who answered two questions for each example. Fig. 5.13 depicts the user interface for LT_2.

5.5.2.3 Audio stimuli

The listening tests used two types of audio stimuli - synthesised using AM and anchors. Specifically, in LT_1, examples in each vocal pair were generated by maximally activating the output neuron and vocal neuron in SVDNet-R1 and SVDNet-R2, respectively. Similarly, examples in each non-vocal pair were generated by minimally activating the output neuron and maximally activating the non-vocal neuron in SVDNet-R1 and SVDNet-R2, respectively. Moreover,

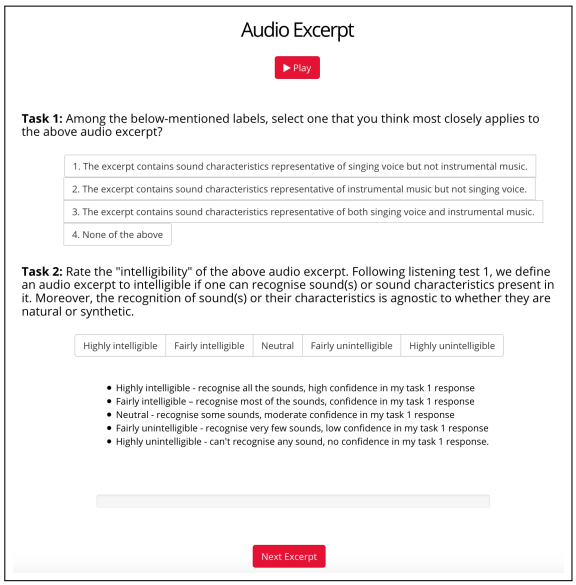

Figure 5.13: The figure presents the user interface for listening test 2.

in LT_1, the ten seeds and hyper-parameters for the AM method are the same as used in 5.4.2.2. This would help in analysing the observations in that section.

In LT_2, the ten synthetic examples were generated by maximally and minimally activating the output neuron in SVDNet-R1 using the five seeds and the best hyper-parameter configurations from Section 5.4.1.2.

Additionally, each listening test used three anchors - a real vocal, a real non-vocal, and a white noise example. The vocal and non-vocal examples are sampled from "03- Say Me Good Bye.mp3" in the Jamendo test dataset and the white noise example was generated using Audacity[4]. Both the tests used the same anchors.

5.5.3 Results

This section presents statistics about the participants and analysis of their responses to the questions in Section 5.5.1.

5.5.3.1 Participants

The perceptual study was open for participation for around three weeks and during that time 74 people consented to participate. Out of the people who consented, 30 participants completed the questionnaire and both the listening tests. This study uses the responses of only these participants. Moreover,

[4]https://www.audacityteam.org

out of these 30 participants, five reported to not use earphones/headphones for the tests. To perform a controlled analysis, the responses from these five participants are not used in further analysis. This also includes a participant who reported hearing impairment and not using headphones. Additionally, the responses from two more participants are not used in further analysis as they responded incorrectly to some of the anchors.

Thus, the analysis of participant responses reported in following sections uses data from 23 participants (18 males and 5 females) with median and mean ages 29.0 years and 30.4 ($\pm$6.7) years, respectively. All 23 participants reported possessing some knowledge of machine learning. Importantly, four participants reported to not possess any musical background, however, as the listening tests mostly required knowledge of high-level and well-known musical concepts (e.g., vocals, non-vocals), the responses from these participants are used in further analysis.

5.5.3.2 Analysis of responses for listening test 1

This section presents the analysis of LT_1 responses that compared the intelligibility of synthetic examples corresponding to the two SVD models. Fig. 5.12 presents the three available choices for each question, Fig. 5.14 depicts the distribution of responses for each pair with examples synthesised using the seeds mentioned on the horizontal axis and Table 5.10 presents the average number of responses for each choice. There are 23 responses (one from each participant) for each pair. The examples in a pair in Plot (A) maximally activate the output and vocal neurons in SVDNet-R1 and SVDNet-R2, respectively. The examples in a pair in Plot (B) minimally and maximally activate the output and non-vocal neurons in SVDNet-R1 and SVDNet-R2, respectively. Sections 5.4.1.2 and 5.4.2.2 suggested that the examples in plots (A) and (B) contain sound(s)/sound characteristics representative of vocals and non-vocals, respectively. Table 5.10 denotes them as vocal and non-vocal examples, respectively.

The results show that on average more participants selected choice 3 that refers to both examples in a pair having similar intelligibility level. Moreover, the Fleiss's Kappa coefficient, that measures the level of agreement between different participants, computed for all the responses across all the examples is 0.23, suggesting a fair agreement between the participants [Fleiss, 1971, Landis and Koch, 1977].

The analysis of response distributions for the vocal and non-vocal examples provides further insights. For the vocal and non-vocal examples, on average, choice 3 and choice 1 recorded the most number of responses, respectively. Thus, the results suggest that there exists a fair agreement that the examples synthe-

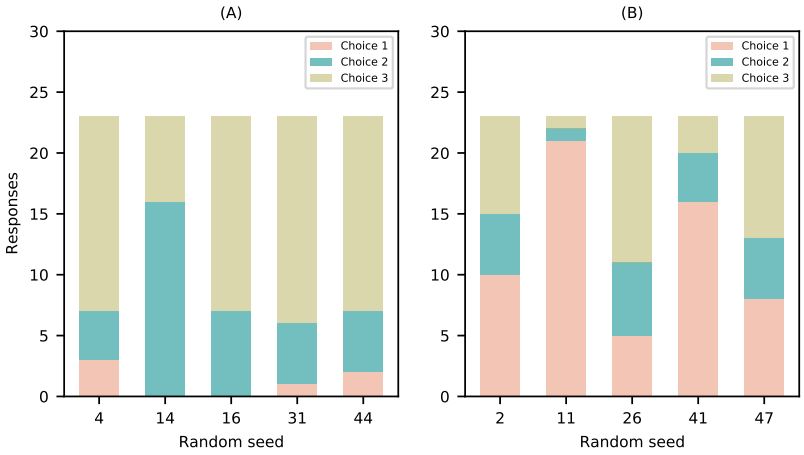

Figure 5.14: The figure depicts response distributions for the first listening test for ten example pairs synthesised using seeds mentioned on the horizontal axis. Plot (A) corresponds to pairs with examples maximally activating the output neuron and the vocal neuron in SVDNet-R1 and SVDNet-R2, respectively. Plot (B) corresponds to pairs with examples minimally activating the output neuron and maximally activating the non-vocal neuron in SVDNet-R1 and SVDNet-R2, respectively. The labels 'choice 1', 'choice 2', and 'choice 3' refer to the three choices available for each question and they refer to example 1 in a pair being more, less, or similarly intelligible as compared to example 2, respectively.

	Choice 1	Choice 2	Choice 3
All examples	6.6(±7.2)	5.8(±3.9)	10.6(±5.8)
Vocal examples	1.2(±1.3)	7.4(±4.9)	14.4(±4.2)
Non-vocal examples	12.0(±6.4)	4.2(±1.9)	6.8(±4.7)

Table 5.10: The table presents the average number of responses for each of the three choices in the listening test 1. The vocal and non-vocal examples represent synthetic examples with vocal and non-vocal sound characteristics. The numbers within the brackets represent standard deviation. The labels 'choice 1', 'choice 2', and 'choice 3' refer to the three choices available for each question and they refer to example 1 in a pair being more, less, or similarly intelligible as compared to example 2, respectively.

sised for the single output neuron in SVDNet-R1 are at least as intelligible as the examples synthesised for the two output neurons in SVDNet-R2.

5.5.3.3 Analysis of responses for listening test 2

LT_2 required the participants to listen to thirteen examples and answer two questions for each one of them. This section presents an analysis of the responses for each of the two questions.

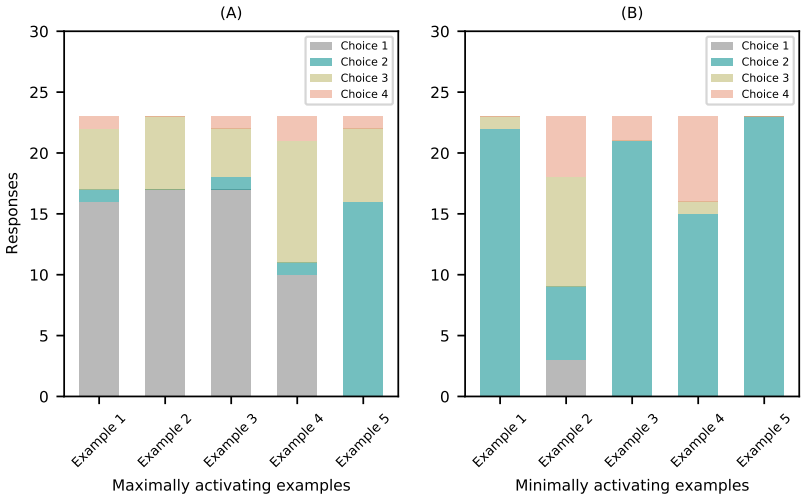

Figure 5.15: The figure depicts the distribution of the responses from the participants for the first question in the second listening test. Plots (A) and (B) present the response distributions for five examples that maximally and minimally activate the output neuron in SVDNet-R1, respectively. The labels 'choice 1', 'choice 2', 'choice 3', and 'choice 4' refer to the four choices available for the first question and they refer to an example containing sound characteristics representative of vocals but not non-vocals, non-vocals but not vocals, both, and none, respectively.

	Choice 1	Choice 2	Choice 3	Choice 4
Maximally activating examples	12.0(±7.3)	3.8(±6.8)	6.2(±2.3)	1.0(±0.7)
Minimally activating examples	0.6(±1.3)	17.4(±7.1)	2.2(±3.8)	2.8(±3.1)

Table 5.11: The table presents the average number of responses for each of the four choices across the two example categories. The maximally and minimally activating examples represent synthetic examples that maximise and minimise the output neuron activation in SVDNet-R1, respectively. The numbers within the brackets represent standard deviation. The labels 'choice 1', 'choice 2', 'choice 3', and 'choice 4' refer to the four choices available for the first question and they refer to an example containing sound characteristics representative of vocals but not non-vocals, non-vocals but not vocals, both, and none, respectively.

Question 1 aimed to understand whether examples that maximally or minimally activate the output neuron in SVDNet-R1 contain sound characteristics corresponding to the vocals and non-vocals, respectively. The participants needed to select one out of the four available choices (see Fig. 5.13). Fig. 5.15 presents the distribution of responses for each of the ten synthetic examples and Table 5.11 presents the average number of responses for each choice across the

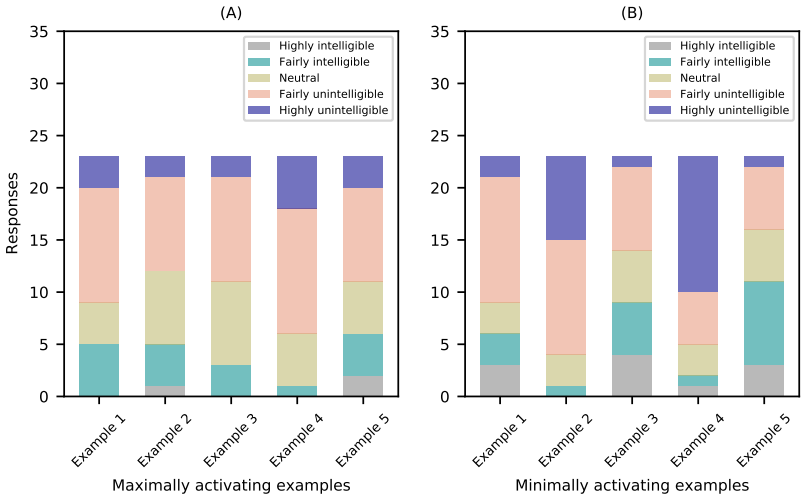

Figure 5.16: The figure depicts the distribution of the responses from the participants for the second question in the second listening test. Plots (A) and (B) present the response distributions for five examples that maximally and minimally activate the output neuron in SVDNet-R1, respectively.

two example categories - maximally activating examples and minimally activating examples.

The results show that on average the participants selected choices 1 and 2 the most for the maximally and minimally activating examples, respectively. This further supports the observation in Section 5.4.1.2 that examples maximally and minimally activating the output neuron in SVDNet-R1 contain sound characteristics representative of vocals and non-vocals, respectively. It is important to note that both choices 1 and 3 correspond to the presence of vocals, thus, on average the number of participants that identified maximally activating examples to contain vocal characteristics is even higher. Interestingly, despite most of the training excerpts containing vocals in the presence of instrumental music, the selection of choice 1 in nearly half of the responses suggests that the neuron has learned some representation of only vocals.

The Fleiss's Kappa coefficients corresponding to the responses for the maximally and minimally activating examples are 0.22 and 0.25, respectively, suggesting fair agreement between the participants. However, if the responses for choices 1 and 3 are combined as they both refer to vocal characteristics, then the Kappa value is 0.65 and 0.57 for the maximally and minimally activating examples, respectively, suggesting that the participants have a borderline substantial agreement.

Question 2 aimed to understand how intelligible examples synthesised

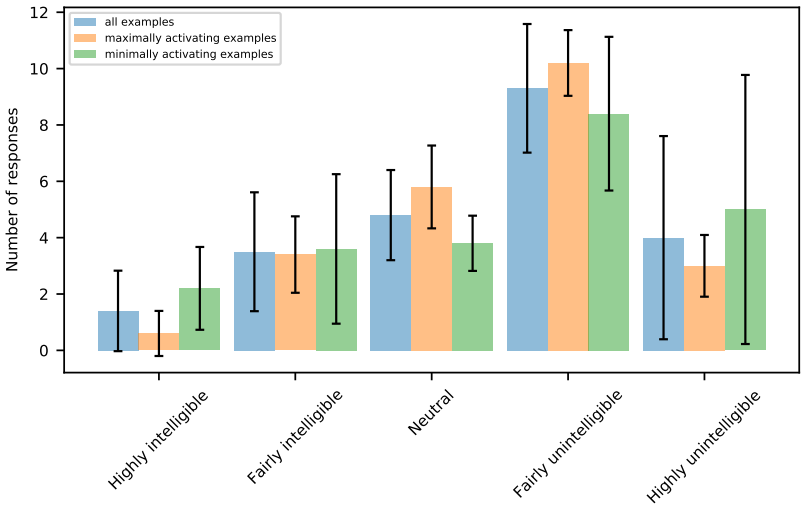

Figure 5.17: The figure depicts the average number of participant responses for each of the five choices mentioned on the horizontal axis. The error bars represent standard deviation.

by the proposed AM method are. Thus, the participants were asked to rate the intelligibility of each of the ten examples from above on a five-point Likert scale [Likert, 1932] from "Highly intelligible" to "Highly unintelligible" (see Fig. 5.13). Fig. 5.16 plots the response distribution across all the examples and Fig. 5.17 plots the average number of responses for each of the five choices.

The results show that on average the participants found the examples to be fairly unintelligible, suggesting that in each example they were able to recognise very few sounds and have low confidence in their answer for question 1. Importantly, there exists variability in participant responses across different examples. This can be understood from the response distributions of the minimally activating examples 4 and 5 in Fig. 5.16 where the choices highly unintelligible and fairly intelligible received the most number of responses, respectively. Moreover, even for the same example, there seems a very low agreement between the participants. For example, in the maximally activating example 2, the number of responses for the choices neutral and fairly unintelligible are quite close. The lack of agreement between the participants becomes further evident by the Kappa coefficient values -0.03, 0.06, and 0.03 for the maximally activating, minimally activating, and all the examples, respectively, suggesting borderline poor agreement. Thus, further analysis would be required to better understand the reasons behind poor agreement between the participants for rating the intelligibility of the synthetic examples.

5.6 Summary and conclusion

This chapter discussed a method to analyse the global behaviour of DNNs. Specifically, the chapter discussed vanilla AM, a method to synthesise examples in the input space by iteratively optimising noise to maximally activate the activations of DNN components (layers, neurons). The chapter presented a new method for AM that uses a GAN as a regulariser in the AM pipeline that aims to keep the synthesised examples realistic. The proposed method performs optimisation in the latent space of the GAN to select a latent code whose corresponding example from the GAN maximally activates the DNN components.

The chapter also proposed a novel method for quantitatively selecting suitable hyper-parameter settings for AM. The method uses the FID score as a metric for automatically evaluating the usefulness of a set of generated explanations. The FID measures the distance between the activation distribution of a set of synthetic examples (from AM) corresponding to a hyper-parameter configuration and the activation distribution of a set of real examples. A lower value of the FID suggests that the two distributions are similar.

The chapter discussed experiments to analyse the effectiveness of the proposed methods. The experiments applied the methods to understand the concepts the output layer neurons encode in two pre-trained deep SVD models. The first SVD model is a state-of-the-art DNN with a single sigmoidal output neuron, while the second SVD model is architecturally similar to the first model, but for its output layer that has two neurons followed by a softmax layer.

The experiments first applied the proposed methods to synthesise examples (mel-spectrograms) that maximally and minimally activate the output layer neuron in the first SVD model. To do that, the experiments first validated whether the hyper-parameter selection method helps in identifying suitable hyper-parameter configurations for AM. The qualitative analysis of the results for 27 configurations suggests that the FID score can effectively rank the configurations in the order of the interpretability of the examples they synthesise. Specifically, the results suggested that the method effectively identifies unsuitable configurations that may generate adversarial examples and suitable configurations that are likely to synthesise realistic examples. The results also suggest that some configurations (with smaller FID and close to the best configuration) may synthesise interpretable examples for some noise vectors. Thus, in addition to selecting the best configuration, it may be helpful to select some other configurations close to the best configuration, but even then the proposed method drastically reduces the hyper-parameter search space.

The chapter further discussed experiments that use the best configurations

from the above experiment to synthesise examples that maximally and minimally activate the single neuron in the first model. The analysis of the results suggests that similar to image classification models, the neuron in SVDNet-R1 is able to capture high-level class concepts. Specifically, examples synthesised to maximise the activation of the output neuron depict the presence of vocal characteristics (e.g., highly harmonic) and the examples synthesised to minimise the output neuron activation depict the presence of non-vocal characteristics (e.g., instrumental onsets).

The chapter also described experiments that apply the proposed methods for hyper-parameter configuration selection and AM to the second deep SVD model (SVDNet-R2) to test if the methods generalise to the new model that is 1.1% less accurate than the first model. The experiments first analysed if the FID score reflects the interpretability of a set of synthesised examples and then used the best configuration corresponding to an output neuron to synthesise examples that maximally activate its pre-softmax activation. The results suggest that both the proposed methods generalise well to the new model. The hyper-parameter selection method effectively identifies the best and worst configurations for both the output neurons. Moreover, the analysis of the examples that the AM method synthesises suggests that each output neuron learns high-level class concepts. Importantly, the two SVD models differ minimally in their architectures, thus, further experiments with models with different architectures would be required to better understand the generalisability of the proposed methods.

The chapter discussed experiments that analysed if examples synthesised for the second model are more interpretable than the examples synthesised for the first model. To do that, the experiment qualitatively compared the synthesised mel-spectrograms for an output neuron in the second model with the synthesised mel-spectrograms for the corresponding setting (minimise or maximise) of the output neuron in the first model. The analysis of results suggests that as compared to the examples from the first model, some examples from the second model visually seem more refined with important input characteristics (onsets, harmonics) more visually prominent. The listening tests also suggest that some examples from the second model are less noisy (more interpretable) than those from the first model. However, it is important to note that these observations do not always hold and overall the single output neuron in the first model seems to learn representations that are fairly similar to the representations learnt by the two neurons in the second model.

Finally, the chapter presented a perceptual study to further analyse some observations from the qualitative analysis of the two SVD models. The study consisted of two listening tests that required the participants to listen to 1.6

second audio examples and answer corresponding questions. The analysis of responses from 23 participants further supports the two observations from the qualitative analysis of the models:

- Maximally and minimally activating examples corresponding to the output neuron in SVDNet-R1 contain sounds with vocal and non-vocal characteristics, respectively.
- The examples synthesised for maximally or minimally activating the output neuron in SVDNet-R1 are at least as intelligible as the corresponding examples for SVDNet-R2.

Importantly, the analysis of responses for the intelligibility level of the examples synthesised by the proposed method suggests that the participants found the examples to be fairly unintelligible, but due to poor agreement between the participants, further analysis would be required to better understand the responses.

Thus, the experiments in this chapter demonstrate that the proposed AM method assists in providing some insights into the global behaviour of machine listening models.

5.7 Reproducibility

The code for all the experiments is available on request. The below mentioned (private) Github repository contains the code, parameters to reproduce the thesis results, and steps to generate new results. Moreover, all the experimental results are available at a public Github repository whose details are mentioned below.

- `https://github.com/saum25/AM_synthesis` includes the code to perform activation maximisation/minimisation using the GAN-based prior and the code to evaluate the hyper-parameter configurations using the FID-based metric.
- `https://github.com/saum25/AM_synthesis_thesis_results` includes all the experimental results from this chapter. Moreover, for each seed used in the experiments, the repository provides both the time-frequency and the temporal representations (after inversion and phase reconstruction) of the synthesised mel-spectrogram.

Chapter 6

Feature inversion

Until now, this thesis discussed methods that help in analysing DNNs by either explaining their predictions (see Chapter 4) or by analysing their components (see Chapter 5). This chapter presents a method that helps in understanding DNNs in both ways. Specifically, the chapter presents feature inversion, a method that analyses DNNs by mapping their latent features back to the input space. The chapter demonstrates the effectiveness of the method by using it to understand a state-of-the-art deep machine listening model (SVDNet) from Chapter 3.

This chapter consolidates the work from two conference publications. The first one showed the effectiveness of feature inversion in explaining predictions from the SVDNet model [Mishra et al., 2018a], while the second one applied feature inversion to understand how the SVDNet model discriminates between the vocal and non-vocal categories [Mishra et al., 2018b]. In addition to the content from the two publications, this chapter includes new figures, plots and a new subsection about analysing the conclusions from Mishra et al. [2018b].

The remainder of this chapter is organised as follows: Section 6.1 highlights how feature inversion complements SLIME and AM methods in understanding deep machine listening models. Section 6.2 introduces feature inversion and presents the methodology for feature inversion. Section 6.3 describes experiments demonstrating the effectiveness of feature inversion in generating instance-wise explanations. Section 6.4 describes experiments employing feature inversion for understanding the latent features of SVDNet. Section 6.5 summarises the key results and highlights the effectiveness of feature inversion in understanding machine listening models. Finally, for reproducibility, Section 6.6 mentions the repositories hosting the source code of all the experiments in this chapter.

6.1 Introduction

As discussed in Chapter 2, researchers have proposed several post-hoc methods to analyse the behaviour of a DNN [Montavon et al., 2018]. One category of methods focuses on explaining predictions of a DNN by identifying input dimensions [Simonyan et al., 2014] or input regions (a group of contiguous dimensions) [Zeiler and Fergus, 2014] that contribute in favour of (or against) a prediction. Chapter 4 discussed why explaining a prediction in terms of input regions is more interpretable for machine listening models. The chapter demonstrated that SLIME tackles some of the key challenges that gradient-based methods face. However, SLIME requires information about the number of interpretable components and a segmentation methodology (e.g., uniformly segmenting input into N time-frequency blocks). Moreover, the explanations that SLIME generates may highlight non-contiguous components (regions), which sometimes makes their interpretation challenging.

Additionally, as Chapter 5 discussed, analysing a DNN by explaining its predictions may help to understand how a model discriminates between the classification categories, but it provides no insights into how a DNN hierarchically maps input features into discriminative representations. Activation maximisation aims to tackle this problem by helping to visualise features that neurons (or a group of neurons) in a DNN have learnt to identify [Yosinski et al., 2015]. However, often for deeper layers, synthesised examples are difficult to interpret due to the multi-faceted nature of neurons [Nguyen et al., 2016b].

This chapter introduces feature inversion and demonstrate its effectiveness to tackle the above challenges. Feature inversion aims to highlight the input content (features) preserved by any layer in a DNN model by inverting the corresponding feature [Mahendran and Vedaldi, 2015]. One of the key contributions of this chapter is to demonstrate that feature inversion can explain DNN predictions, making it a versatile method that can explain a DNN as well as its predictions. Using feature inversion to explain predictions tackles the challenges that SLIME faces as the proposed method does not need any auxiliary information and will generate contiguous regions as explanations[1]. Moreover, the method generates an explanation significantly more quickly than SLIME.

The other key contribution of this chapter is to apply feature inversion to understand how a state-of-the-art machine listening model [Schlüter and Grill, 2015] discriminates between the classification categories. The SVDNet model is a binary classifier that classifies a mel-spectrogram excerpt of around 1.6 seconds duration into 'vocal' or 'non-vocal' categories. Chapter 3 described the

[1] [Ribeiro et al., 2016b] earlier proposed homologous "super-pixel"-based explanations for images.

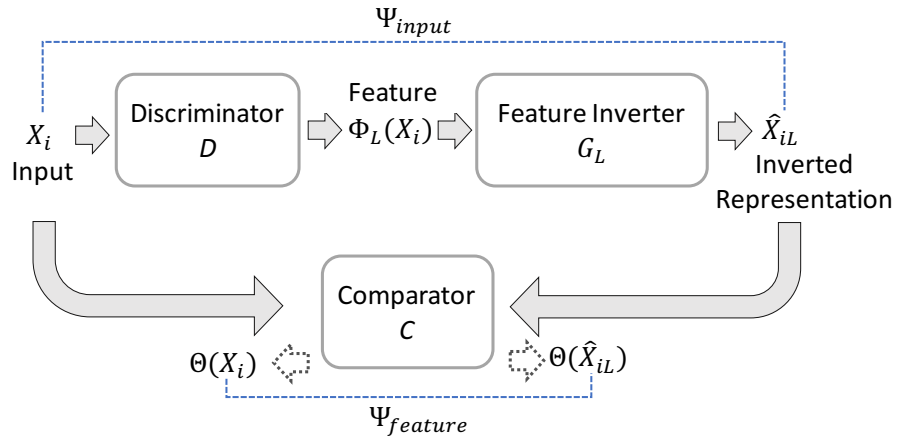

Figure 6.1: Functional block diagram of the feature inversion method. The method inverts a feature $\Phi_L(\mathbf{x}_i)$ from a layer L by training a feature inverter G_L that jointly minimises the input space loss Ψ_{input} and the feature space loss $\Psi_{feature}$. Φ_L and Θ are the representation functions of a discriminator D and comparator C, respectively.

architecture, training, and evaluation of the SVDNet model.

6.2 Methodology

Feature inversion aims to map the feature generated at any layer of a DNN back to a plausible input. Each layer in a DNN maps an input feature to an output feature and in the process ignores the input content that does not seem relevant to the classification task. Thus, the inversion of a feature from any layer of a DNN will highlight the input content preserved by that layer.

Fig. 6.1 provides an overview of the feature inversion method this chapter uses. The method uses the approach of Dosovitskiy and Brox [2016a] but with some modifications. Their method inverts features from a DNN layer by training an up-convolutional neural network (this thesis calls it a 'feature inverter'). The method trains this feature inverter by minimising the input space loss Ψ_{input}, defined as the squared Euclidean distance between an input image and its inverted representation. Although this approach has some benefits, the reconstructions it generates are blurry due to the effect of input averaging (see Chapter 2). Hence, to reduce the effect of input averaging the feature inversion method in this chapter modifies the loss function of the method proposed by Dosovitskiy and Brox [2016a].

Recent works demonstrate that minimising loss in the perceptual space helps to reduce the over-smoothness problem for image generation models [Dosovitskiy

and Brox, 2016b, Snell et al., 2017]. This chapter extends this idea to machine listening. Thus, in addition to the input space loss Ψ_{input}, the feature inversion method also calculates the feature space loss $\Psi_{feature}$. The method defines the total loss Ψ as:

$$\Psi = \lambda_{input}\Psi_{input} + \lambda_{feature}\Psi_{feature} \tag{6.1}$$

where λ_{input} and $\lambda_{feature}$ weight the losses of the input space and feature space. Thus, the method trains a feature inverter (up-convolutional neural network) G_L to invert a feature $\Phi_L(\mathbf{x}_i)$ by jointly minimising the input space and feature space losses. To evaluate $\Psi_{feature}$, the method uses the approach from Dosovitskiy and Brox [2016b] who use a comparator C to map an input $\mathbf{x}_i$ and its inverted representation $\hat{\mathbf{x}}_{iL}$ to the feature space. A comparator is a discriminative model that may be pre-trained or learned and may or may not be of the same depth as the discriminator D (the model whose features the feature inversion method inverts). The feature inversion method can even use D as a comparator by extracting feature vectors at a layer of D (e.g., Dosovitskiy and Brox [2016b] use the deepest convolutional layer of AlexNet as a comparator for inverting AlexNet).

Formally, given an input excerpt $\mathbf{x}_i \in \mathbb{R}^n$ and a representation function $\Phi_L : \mathbb{R}^n \rightarrow \mathbb{R}^d$ that maps $\mathbf{x}_i$ to a d-dimensional feature $\Phi_L(\mathbf{x}_i)$ at a layer L of a discriminator D, the feature inversion method trains a feature inverter G_L that maps $\Phi_L(\mathbf{x}_i)$ to an inverted representation $\hat{\mathbf{x}}_{iL} \in \mathbb{R}^n$. In order to do that, the method calculates Ψ_{input} and $\Psi_{feature}$. Given a comparator C with a representation function $\Theta : \mathbb{R}^n \rightarrow \mathbb{R}^{d'}$, the method defines $\Psi_{feature}$ as the squared Euclidean distance between $\Theta(\mathbf{x}_i)$ and $\Theta(\hat{\mathbf{x}}_{iL})$, where $\hat{\mathbf{x}}_{iL}$ is an inverted representation for an input $\mathbf{x}_i$ at layer L and d' is the dimensionality of the feature space for C. Similarly, the method defines Ψ_{input} as the squared Euclidean distance between $\mathbf{x}_i$ and $\hat{\mathbf{x}}_{iL}$. The method trains an up-convolutional neural network $G_L(\Phi_L(\mathbf{x}_i); \mathbf{w})$ with parameters $\mathbf{w}$ by the optimisation

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_i (\|\mathbf{x}_i - G_L\{\Phi_L(\mathbf{x}_i); \mathbf{w}\}\|_2^2 + \|\Theta(\mathbf{x}_i) - \Theta(G_L\{\Phi_L(\mathbf{x}_i); \mathbf{w}\})\|_2^2) + \gamma \cdot \|\mathbf{w}\|_2^2 \tag{6.2}$$

where $\gamma > 0$ is the regularisation constant. Once the method trains G_L, it can invert any feature $\Phi_L(\mathbf{x}_i)$ by a forward pass through G_L:

$$\hat{\mathbf{x}}_{iL} = G_L(\Phi_L(\mathbf{x}_i); \mathbf{w}^*) \tag{6.3}$$

Algorithm 3 presents the pseudocode of training an L_{th} layer feature inverter G_L. The following sections describe two experiments that use the above feature

Algorithm 3: Training an L_{th} layer feature inverter G_L

Input: Discriminator D with the L_{th} layer representation function Φ_L, comparator C with representation function Θ, and an L_{th} layer feature inverter G_L with initial parameters $\mathbf{w}_0$

Input: Input batch size m, number of training batches t, number of epochs N, regularisation constant γ, input and feature space loss coefficients λ_{input} and $\lambda_{feature}$, respectively, Adam hyper-parameters - initial learning rate α, β_1, β_2, ϵ

Output: Final G_L parameters $\mathbf{w}^*$

for *epoch* $\in \{1, 2, 3, ..., N\}$ **do**
  Randomly sample t batches, each with m inputs;
  for *batch index* $b \in \{1, 2, 3, ..., t\}$ **do**
    Select inputs $\{\mathbf{x}_i\}_{i=1}^{m}$ corresponding to the index b;
    for *input index* $i \in \{1, 2, 3, ..., m\}$ **do**
      $\hat{\mathbf{x}}_{iL} \leftarrow G_L\{\Phi_L(\mathbf{x}_i); \mathbf{w}\}$;
      $\Psi^i_{input} \leftarrow \|\mathbf{x}_i - \hat{\mathbf{x}}_{iL}\|_2^2$;
      $\Psi^i_{feature} \leftarrow \|\Theta(\mathbf{x}_i) - \Theta(\hat{\mathbf{x}}_{iL})\|_2^2$;
      $L^{(i)} \leftarrow \lambda_{input}\Psi^i_{input} + \lambda_{feature}\Psi^i_{feature}$;
    end for
    $\mathbf{w} \leftarrow \text{Adam}(\nabla_{\mathbf{w}}(\frac{1}{m}\sum_{i=1}^{m} L^{(i)} + \gamma \cdot \|\mathbf{w}\|_2^2), \mathbf{w}, \alpha, \beta_1, \beta_2, \epsilon)$;
  end for
  $\mathbf{w}^* \leftarrow \mathbf{w}$;
end for
return $\mathbf{w}^*$;

inversion method. The first experiment demonstrates that feature inversion generates effective explanations for DNN predictions. The second experiment uses feature inversion to analyse latent features of a state-of-the-art deep machine listening model.

6.3 Explaining DNN predictions using feature inversion

This section proposes and demonstrates a novel method for explaining DNN predictions. The method uses feature inversion to locate a region in an input that contributes the most to a prediction. This section first explains the explanation generation method. Then, it describes experiments that involve applying the proposed method to explain predictions of SVDNet.

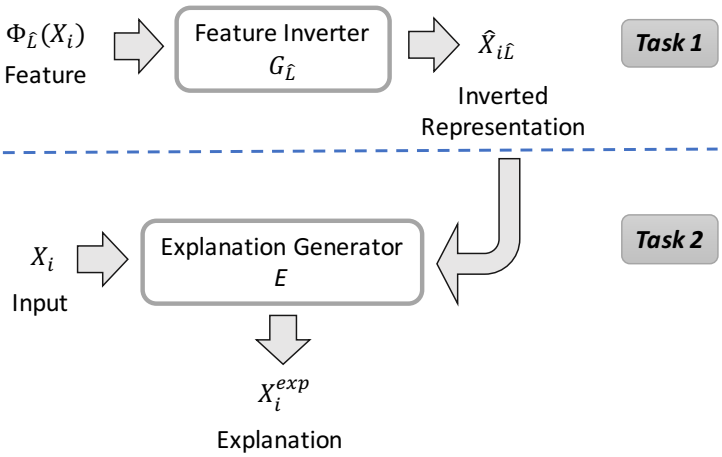

Figure 6.2: Functional block diagram of the explanation generation step. Task 1 involves using a feature inverter $G_{\hat{L}}$ to invert a feature $\Phi_{\hat{L}}(\mathbf{x}_i)$ from the deepest hidden layer $\hat{L}$. Task 2 involves using an explanation generator E and an inverted representation $\hat{\mathbf{x}}_{i\hat{L}}$ to generate an explanation $\mathbf{x}_i^{exp}$ for the categorisation of input $\mathbf{x}_i$.

6.3.1 Intuition

It is conceivable that a DNN predicts (classifies) by using latent features from its deepest hidden layer. A DNN generates these features automatically from inputs and there is limited understanding about what those features represent. However, inversion of a feature from the deepest hidden layer will highlight a region in the input space that a DNN considers important for prediction, suggesting that the discriminative feature is prominently present in the highlighted region of input. For example, if an image recognition model classifies an image to the 'Cat' class by looking at the face of the cat (assumption), then an inverted representation from the deepest hidden layer feature should highlight the cat's face more prominently (explanation) than other image components. Thus, to explain a DNN prediction, the explanation method proposes to invert a feature at its deepest hidden layer and convert the inverted representation to a binary mask that masks the input. The unmasked section in the input will highlight a region that influences a model prediction.

6.3.2 Explanation generation method

The proposed explanation method consists of two steps.

- Step 1 involves training a feature inverter $G_{\hat{L}}$ that maps a feature $\Phi_{\hat{L}}(\mathbf{x}_i)$ from the deepest hidden layer $\hat{L}$ to an inverted representation $\hat{\mathbf{x}}_{i\hat{L}}$.

Algorithm 4: The explanation generation step

Input: Pre-trained discriminator D with the deepest layer representation function $\Phi_{\hat{L}}$ and pre-trained feature inverter $G_{\hat{L}}$ with parameters $\mathbf{w}^*$ for inverting the deepest layer $\hat{L}$ in D

Input: Input $\mathbf{x}_i$ and thresholding constant α_{th}

Output: Explanation $\mathbf{x}_i^{exp}$ for the prediction of $\mathbf{x}_i$

$\hat{\mathbf{x}}_{i\hat{L}} \leftarrow G_{\hat{L}}(\Phi_{\hat{L}}(\mathbf{x}_i); \mathbf{w}^*)$; `// generate inverted representation` $\hat{\mathbf{x}}_{i\hat{L}}$

$\mathbf{x}_i^b \leftarrow normalise\_and\_threshold(\hat{\mathbf{x}}_{i\hat{L}}, \alpha_{th})$; `// binary mask`

$\mathbf{x}_i^{exp} \leftarrow \mathbf{x}_i \odot \mathbf{x}_i^b$;

return $\mathbf{x}_i^{exp}$;

- Step 2 is the explanation generation step. It involves using $G_{\hat{L}}$ to explain why D (a DNN) classified $\mathbf{x}_i$ to the category C. Fig. 6.2 depicts the two tasks in the explanation generation step. Task 1 generates an inverted representation $\hat{\mathbf{x}}_{i\hat{L}}$ using $G_{\hat{L}}$ by

$$\hat{\mathbf{x}}_{i\hat{L}} = G_{\hat{L}}(\Phi_{\hat{L}}(\mathbf{x}_i); \mathbf{w}^*) \tag{6.4}$$

  Task 2 involves feeding an input excerpt $\mathbf{x}_i$ and its inverted representation $\hat{\mathbf{x}}_{i\hat{L}}$ to an explanation generator E that generates an explanation. E firstly normalises $\hat{\mathbf{x}}_{i\hat{L}}$ to the range $[0, 1]$ and then thresholds the normalised output using a thresholding constant α_{th} generating a binary mask $\mathbf{x}_i^b \in \{0, 1\}^n$. If a normalised bin value is less than α_{th} then the method sets it to 0 else sets it to 1. The method uses this thresholding approach as it seems reasonable to assume that magnitude of a bin in the inverted representation relates to its importance in the discrimination task. Then, E generates an explanation $\mathbf{x}_i^{exp}$ by masking $\mathbf{x}_i$.

$$\mathbf{x}_i^{exp} = \mathbf{x}_i \odot \mathbf{x}_i^b \tag{6.5}$$

  where $\odot$ refers to element-wise multiplication. The non-zero (unmasked) dimensions in $\mathbf{x}_i^{exp}$ highlight the input region influencing the prediction. Algorithm 4 presents the pseudocode of the explanation generation step. The pseudocode for training $G_{\hat{L}}$ can be obtained by replacing L by $\hat{L}$ in Algorithm 3.

This section now describes an experiment that demonstrates the proposed explanation generation method to explain predictions of SVDNet, a state-of-the-art deep SVD model. To do that, the experiment first trains a feature inverter for the FC8 layer. Later, it uses the feature inverter to explain SVDNet predictions. Finally, it qualitatively and quantitatively verifies the reliability of

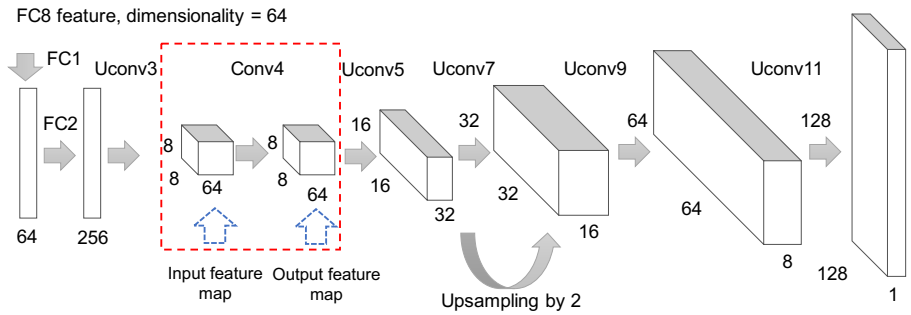

Figure 6.3: An overview of the model architecture for inverting features from the FC8 layer of SVDNet. FC, Conv and UConv refer to the fully-connected, convolutional and up-convolutional layers, respectively. The highlighted components represent the 'Conv4' convolutional layer and its input and output feature maps. Due to space restrictions, the figure shows only one convolutional layer.

the generated explanations.

6.3.3 Feature inverter architecture

The experiment trains a feature inverter $G_{\hat{L}}$ (an up-convolutional neural network [Dosovitskiy and Brox, 2016a]) to invert features from the FC8 layer (deepest hidden layer) of SVDNet. The architecture of AlexNet-FC8 feature inverter in Dosovitskiy and Brox [2016a] inspires the design of the feature inverter in this section, but the experiment adapts the architecture to suit SVDNet.

The feature inverter inverts a 64-dimensional feature by systematically upsampling it. $G_{\hat{L}}$ uses 'up-convolutional layers' (also referred to as 'deconvolutional layers' or 'transposed convolutional layers') [Dosovitskiy et al., 2015, Dosovitskiy and Brox, 2016a, Dosovitskiy et al., 2017] to perform upsampling and strided convolution using 4×4 filters with 2×2 stride and 1×1 cropping. This configuration of up-convolutional layers (this chapter calls them 'UConv') upsamples an input feature map by 2.

The experiment also adds one convolutional layer (Conv) after every UConv layer, as suggested in Dosovitskiy et al. [2017]. This increases model capacity and generates visually improved inverted representations. The convolutional layers perform convolutions using 3×3 filters with 1×1 stride and 1×1 zero-padding.

The network uses batch normalisation layers to make sure the input to each layer follows a standard normal distribution [Ioffe and Szegedy, 2015]. The network employs exponential linear unit (ELU) non-linearities ($y(x) = (x > 0)?x : e^x - 1$) to process the output of each neuron [Clevert et al., 2016]. $G_{\hat{L}}$ generates an output of 128×128 spatial size and later crops it to match the input

Layer	Input shape	Units	Filter	Stride	Output shape
FC1	64×1	64	-	-	64×1
FC2	64×1	256	-	-	256×1
Reshape	256×1	-	-	-	$16 \times 4 \times 4$
UConv3	$16 \times 4 \times 4$	64	4×4	2×2	$64 \times 8 \times 8$
Conv4	$64 \times 8 \times 8$	64	3×3	1×1	$64 \times 8 \times 8$
UConv5	$64 \times 8 \times 8$	32	4×4	2×2	$32 \times 16 \times 16$
Conv6	$32 \times 16 \times 16$	32	3×3	1×1	$32 \times 16 \times 16$
UConv7	$32 \times 16 \times 16$	16	4×4	2×2	$16 \times 32 \times 32$
Conv8	$16 \times 32 \times 32$	16	3×3	1×1	$16 \times 32 \times 32$
UConv9	$16 \times 32 \times 32$	8	4×4	2×2	$8 \times 64 \times 64$
Conv10	$8 \times 64 \times 64$	8	3×3	1×1	$8 \times 64 \times 64$
UConv11	$8 \times 64 \times 64$	1	4×4	2×2	$1 \times 128 \times 128$
Slice	$1 \times 128 \times 128$	-	-	-	$1 \times 115 \times 80$

Table 6.1: The architecture of the FC8 feature inverter. Input and output shapes are ordered as: number of channels × time × frequency. FC, Conv and UConv refer to the fully-connected, convolutional and up-convolutional layers, respectively. Units refers to the number of filters in a Conv or UConv layer or the number of neurons in an FC layer.

excerpt size (115×80). The number of learnable parameters for this network is 131201. Fig. 6.3 depicts an overview of the feature inverter architecture and Table 6.1 provides additional details.

6.3.4 Feature inverter training

The experiment trains the feature inverter by using 64-dimensional FC8 features. SVDNet extracts one feature per input audio excerpt (normalised log-scaled mel-spectrogram of about 1.6 sec, see section 3.4.1). The experiment generates audio excerpts from the Jamendo training dataset using a hop size of 10 frames (140 ms). Fig. 6.4 shows two audio excerpts, one from each classification category, from the Jamendo test dataset. The experiment does not use any of the data augmentation methods from Schlüter and Grill [2015] and trains the feature inverter on a dataset size of about $100k$ features (vectors).

The experiment uses SVDNet truncated at the Conv5 layer as the comparator, i.e., it encodes the input excerpt and its inverted representation using the Conv5 features. The experiment initialises the feature inverter weights using He normal initialisation [He et al., 2015]. For a mini-batch of 32 randomly selected excerpts, the training objective jointly minimises the input and feature space losses using Eq. 6.2 and updates model parameters using Adam [Kingma and Ba, 2015]. Table 6.1 shows the input and output shapes without the batch size dimension. During training the input and output shapes in Table 6.1 are extended by batch size as an additional dimension. The experiment sets the scaling factors $\lambda_{input} = \lambda_{feature} = 1$ (See Eq. 6.1). The training procedure starts training with an initial learning rate of 0.001 and decreases it by 50%

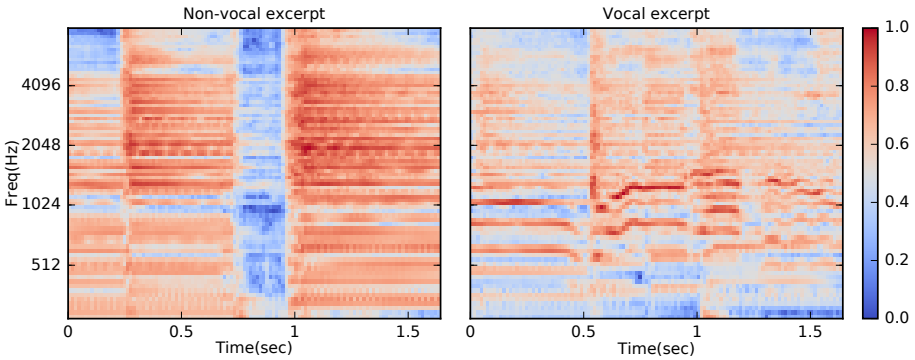

Figure 6.4: Normalised log-scaled mel-spectrogram excerpts from the song "03 - Say me Good Bye.mp3" in the Jamendo test dataset. The left subplot depicts an excerpt (time index : 1.43 seconds - 3.03 seconds) from the non-vocal category. The right subplot depicts an excerpt (time index : 33.00 seconds - 34.65 seconds) from the vocal category.

when the training loss does not change for 2 consecutive epochs.

To prevent overfitting, the experiment uses L2 weight decay with $\gamma = 10^{-4}$ (See Eq. 6.2) and runs the optimisation to a fixed number of weight updates (30 epochs). The experiment selects a feature inverter that gives the lowest loss on the Jamendo validation subset. It is important to note that the hyper-parameters and training methodology were identified using two approaches - by performing grid search in the hyper-parameter space and by adopting some values from previous works. For example, to decide the SVDNet layer features to be used for computing the feature space loss, grid search was performed using the SVDNet layer features. Similarly, suitable scaling factors and learning rate were identified using grid search. However, the mini-batch size and the approach to train the model for a fixed number of weight updates were used from Schlüter and Grill [2015].

6.3.5 Instance-wise explanations for SVDNet

This section uses the trained feature inverter $G_{\hat{L}}$ to explain SVDNet predictions using the methodology described in Section 6.3.2. Fig. 6.5 shows visualisations from the explanation generation step for an excerpt from the Jamendo test dataset. SVDNet correctly classifies this excerpt to the 'non-vocal' category with 95.16% confidence. The experiment uses $G_{\hat{L}}$ to identify the input region influencing this prediction. The explanation method first normalises and thresholds the inverted representation (from the FC8 feature) to generate a binary mask using $\alpha_{th} = 0.7$. Then, it generates the explanation by applying the mask to the input mel-spectrogram excerpt. The explanation suggests that the

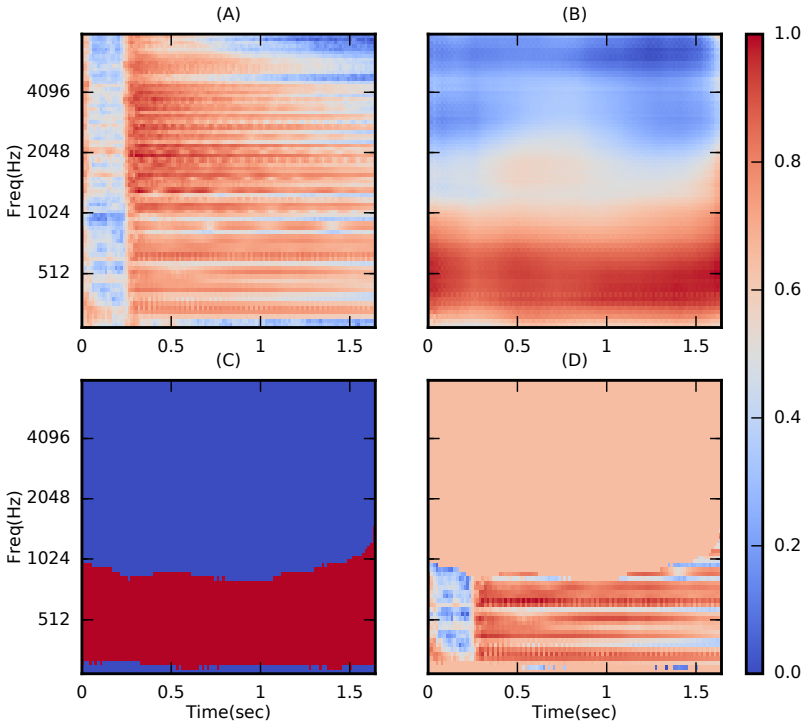

Figure 6.5: Visualisations depicting the explanation generation steps for a randomly selected non-vocal excerpt (time index : 10.00 seconds - 11.65 seconds) from the song "03 - Say me Good Bye.mp3" in the Jamendo test dataset. (A) input mel-spectrogram, (B) inverted representation, (C) binary mask using $\alpha_{th} = 0.7$, and (D) explanation.

information in the lower frequency region (approximately between 150 Hz and 1024 Hz) influences the prediction of the selected excerpt.

We can use instance-wise explanations to verify the trustworthiness of a model. For example, consider the instance shown in Fig. 6.6 (A) that has the vocal and non-vocal sections located in non-overlapping temporal segments (about 0.9 sec singing voice followed by about 0.7 sec instrumental music). SVD-Net classifies this instance to the vocal category with 80% confidence. Fig. 6.6 (B) depicts the explanation (using $\alpha_{th} = 0.7$) for this prediction. The explanation shows that frequencies in the range 300 Hz - 1500 Hz in the first one second of the excerpt contribute most to the classifier's prediction. Similarly, the instance in Fig. 6.6 (C) has about 0.4 sec instrumental music followed by about 1.1 sec singing voice. SVDNet classifies this instance to the vocal category with 98% confidence. The explanation in Fig. 6.6 (D) (using $\alpha_{th} = 0.7$)

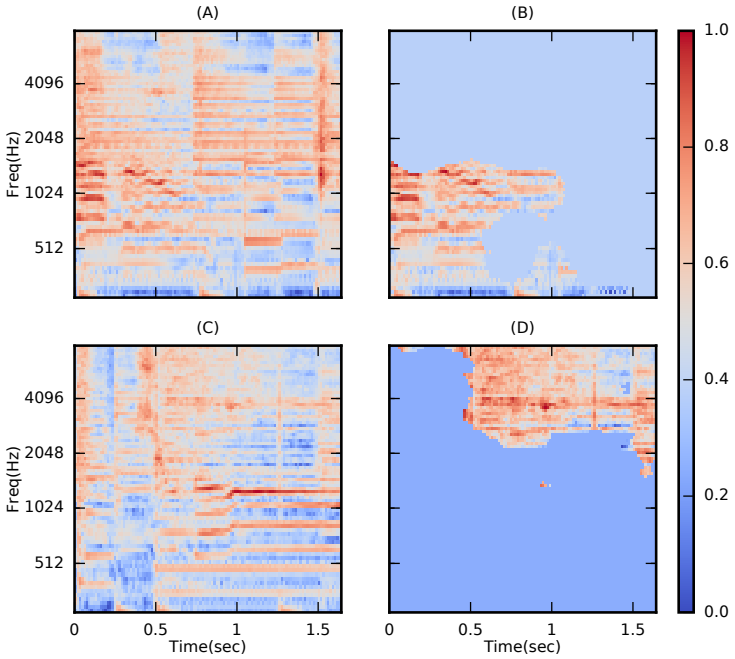

Figure 6.6: Explanations for two excerpts from the song "03 - Say me Good Bye.mp3" in the Jamendo test dataset with the 'vocal' and 'non-vocal' categories as separate temporal segments. (A) represents the input excerpt from 34.00 seconds - 35.65 seconds (confidence score = 0.80) and (B) its explanation. Similarly, (C) represents the input excerpt from 112.00 seconds - 113.65 seconds (confidence score = 0.98) and (D) its explanation.

shows that frequencies in the range 2048 Hz - 8000 Hz in the last one second of the excerpt contribute most to the prediction. In other words, in both cases, the model is using information from the vocal segments to categorise inputs to the vocal category. Such an understanding assists in gaining trust in a model's predictions.

6.3.6 Quantitative evaluation of the proposed method

Researchers often verify their explanation methods using qualitative approaches. For example, an explanation method 'A' is better than a method 'B' if the former generates comparatively cleaner saliency maps. However, recent works have stressed the importance of the quantitative evaluation of explanation methods [Samek et al., 2017, Doshi-Velez and Kim, 2017]. This section reports the results of the quantitative evaluation of the explanation method. The experiment evaluates the method by adapting the region perturbation method from Samek et al. [2017] to suit the region-based explanations. The experiment hypothesis that if an explanation method accurately identifies the influential region in an input,

then masking (removing) the remaining input, should not affect the prediction. It means that the prediction probability may change, but the prediction label should remain the same. Thus, for each input excerpt, the evaluation method feeds the masked representation $\mathbf{x}_i^{exp}$ (see Equation 6.5) to SVDNet and checks whether this modifies the prediction label for the input.

Formally, for a set of N randomly selected input excerpts, the evaluation method considers the initial class labels assigned by SVDNet as the ground truth. It then feeds the masked representations, one from each input, and calculates the number of excerpts N_E for which SVDNet changes its predictions (this chapter calls this approach 'M_1'). Importantly, the evaluation method considers initial model predictions irrespective of their category (e.g., vocal) or their prediction type (e.g., true positive). The evaluation method defines a metric that quantifies how accurately the explanation method identifies input regions that are the most influential to model predictions. The metric, percentage explanation loss is given by $E_{loss} = 100 \cdot \left(\frac{N_E}{N}\right)$. For example, say for $N = 4$ excerpts the sequence of predictions from SVDNet is $(0, 1, 1, 0)$, where 0 and 1 correspond to the 'non-vocal' and 'vocal' categories, respectively. If after feeding the masked versions of inputs to SVDNet, the new sequence of predictions is $(0, 0, 1, 0)$, then, $N_E = 1$ and $E_{loss} = 25\%$.

The experiment also evaluates the explanation method using an approach 'M_2' that for every input $\mathbf{x}_i$ first inverts the binary mask $\mathbf{x}_i^b$ (see Section 6.3.2) by swapping ones with zeros and vice-versa and then uses the inverted mask to generate an SVDNet prediction as discussed above. Such an analysis helps in understanding the influence of input regions that the explanation method identifies as non-important towards model predictions.

Fig. 6.7(A) reports the quantitative evaluation results for a set of 5868 randomly chosen excerpts from the Jamendo test dataset. The experiment creates the set by first randomly selecting a reading offset between the time indices 5 seconds and 20 seconds and then starting from the reading offset it selects around 200 excerpts per audio file with a hop size of 0.5 seconds. The 'red' and 'green' line plots depict the change in % E_{loss} with uniform variations in the masking threshold for M_1 and M_2 approaches, respectively. The subplot (A) shows that for $M_1^{Jamendo}$, the explanation loss gradually increases from 0% ($\alpha_{th} = 0$, no input masking) to a maximum value of 58.41% ($\alpha_{th} = 1$, full input masking). Moreover, for $\alpha_{th} \leq 0.5$, the explanation loss is $\leq 15.89\%$ suggesting that the explanation method identifies influential input regions with a fair degree of accuracy. The variation in % E_{loss} for $M_2^{Jamendo}$ is nearly the opposite of $M_1^{Jamendo}$. For example, for $\alpha_{th} \leq 0.5$, the explanation loss is $\geq 34.16\%$ suggesting that feeding input regions that the explanation method identifies as

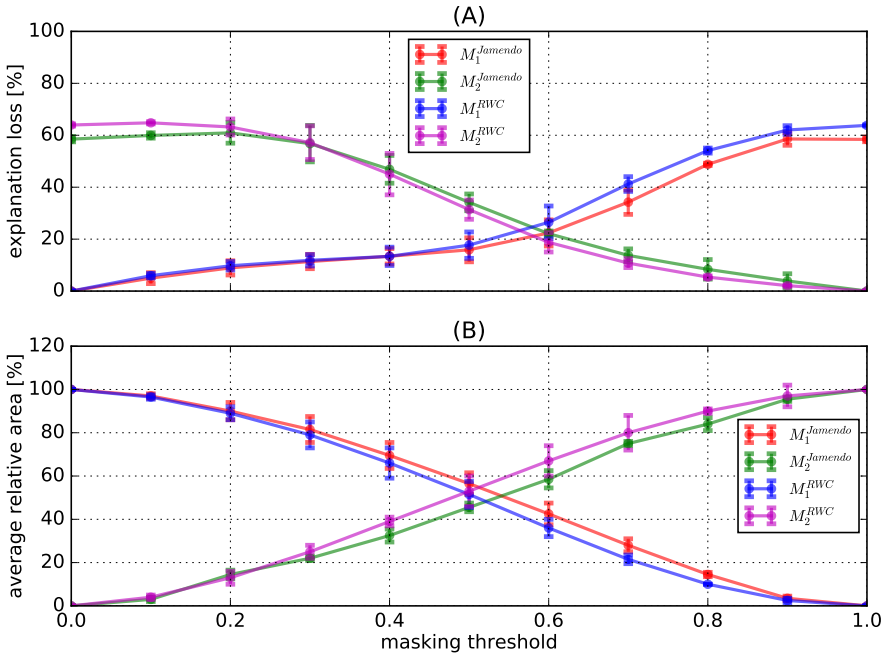

Figure 6.7: Quantitative evaluation of the proposed explanation method for a set of randomly selected instances from the Jamendo and RWC test datasets. For uniform changes in the masking threshold, subplots (A) and (B) depict variations in the % explanation loss and % average relative area of explanations, respectively. M_1 and M_2 are two approaches to transform inputs fed to SVDNet. The error bars represent standard deviation.

non-important flips predictions for a larger number of inputs.

Fig. 6.7(A) also shows that the choice of α_{th} influences the loss resulting from an explanation. For example, for $M_1^{Jamendo}$, $\alpha_{th} \geq 0.7$ results in less accurate explanations ($E_{loss} \geq 34.24\%$), while $\alpha_{th} \leq 0.4$ generates fairly accurate ($E_{loss} \leq 13.41\%$) explanations. To decide an appropriate value of α_{th}, the experiment proposes to use the relative area of an explanation as an additional constraint. The experiment defines the relative area of an explanation as the ratio of the number of unmasked bins to the total number of bins. In Fig. 6.7(B), for the same input excerpts that the experiment uses for Fig. 6.7(A), the experiment plots the variation in average relative area as α_{th} changes uniformly. The experiment uses this information to select a suitable α_{th}. For $M_1^{Jamendo}$, $\alpha_{th} = 0.5$ seems a fair choice as the explanations have % average relative area $= 56.50\%$ and are fairly accurate (% explanation loss is 15.89%).

The experiment also extends the evaluation of the explanation method to the RWC dataset (see Section 3.2). The RWC dataset is not pre-split into a separate test set. Thus, the experiment creates a test set by randomly sampling

20 audio files from which it randomly samples 7966 excerpts using the same methodology as for the Jamendo dataset. The experiments aims to analyse if the quantitative evaluation results from Jamendo generalise to RWC. Fig. 6.7(A) depicts the evaluation results for the RWC dataset for both M_1 and M_2 approaches. For lower values of α_{th} (≤ 0.5), the explanation method performs similarly on RWC, but for higher values of α_{th}, E_{loss} increases on an average by 5.04% for M_1^{RWC}. It seems reasonable to believe that it due to the training of both the discriminator (SVDNet) and feature inverter only on the Jamendo dataset. This results in higher reconstruction error between an input and its inverted representation for the RWC dataset, leading to less accurate explanations. Thus, an accurate inversion model is crucial to achieving low explanation loss.

For M_2^{RWC}, the plot of change in explanation loss for uniform changes in the masking threshold is similar to the one for Jamendo with some minor variations. For $\alpha_{th} \leq 0.3$, the explanation loss for RWC is higher than Jamendo on an average by 3.24%, and for $\alpha_{th} \geq 0.4$ the explanation loss on RWC is lower than Jamendo on an average by 2.25%.

Fig. 6.7(B) also depicts that the line plot of average relative area for RWC is similar in shape to Jamendo for both M_1 and M_2 approaches. Moreover, the explanations for M_1^{RWC} are on an average 2.81% smaller than for $M_1^{Jamendo}$. Finally, for the best threshold value from the Jamendo dataset ($\alpha_{th} = 0.5$), RWC generates slightly less accurate ($E_{loss} = 17.69\%$) explanations with % average relative area = 51.5%. Hence, the results from the quantitative analysis confirm that for a suitable masking threshold, the explanation method can successfully recognise an input region that is influential to a classifier's prediction.

6.4 Understanding SVDNet features

This section describes experiments that use the feature inversion method from Section 6.2 for understanding the features that each layer of SVDNet extracts from any input and to gain insight into how SVDNet classifies an input to the 'vocal' or 'non-vocal' categories. To do that, the experiments train eight feature inverters, one per layer of the SVD model. This section first explains the architectures and training details of the feature inverters. Then, the section discusses experiments that quantify the performance of the feature inverters on two datasets by calculating the normalised reconstruction error (NRE) defined as the normalised Euclidean distance between an input and its inverted representation. Finally, the section discusses experiments that qualitatively analyse the inverted features for both the classification categories to understand the preserved input content at each layer of the model. Similarly, the experiments

Layer	Input shape	Units	Filter	Stride	Output shape
FC1	256×1	256	-	-	256×1
Reshape	256×1	-	-	-	$16 \times 4 \times 4$
Uconv2	$16 \times 4 \times 4$	64	4×4	2×2	$64 \times 8 \times 8$
Conv3	$64 \times 8 \times 8$	64	3×3	1×1	$64 \times 8 \times 8$
Uconv4	$64 \times 8 \times 8$	32	4×4	2×2	$32 \times 16 \times 16$
Conv5	$32 \times 16 \times 16$	32	3×3	1×1	$32 \times 16 \times 16$
Uconv6	$32 \times 16 \times 16$	16	4×4	2×2	$16 \times 32 \times 32$
Conv7	$16 \times 32 \times 32$	16	3×3	1×1	$16 \times 32 \times 32$
Uconv8	$16 \times 32 \times 32$	8	4×4	2×2	$8 \times 64 \times 64$
Conv9	$8 \times 64 \times 64$	8	3×3	1×1	$8 \times 64 \times 64$
Uconv10	$8 \times 64 \times 64$	1	4×4	2×2	$1 \times 128 \times 128$
Slice	$1 \times 128 \times 128$	-	-	-	$1 \times 115 \times 80$

Table 6.2: The architecture of the FC7 feature inverter. Input and output shapes are ordered as: number of channels $\times$ time $\times$ frequency. FC, Conv and UConv refer to the fully-connected, convolutional and up-convolutional layers, respectively. Units refer to the number of filters in a Conv or UConv layer or the number of neurons in an FC layer.

analyse the inverted features for inputs selected from different datasets to test whether the conclusions from one dataset generalise to the other.

6.4.1 Feature inverter architectures

The experiments train eight up-convolutional neural networks (feature inverters), one per layer of SVDNet, to invert the features generated by it. Section 6.3.3 discussed in detail the architecture of the FC8 feature inverter. This section discusses the architectures of the other feature inverters. The experiments design two categories of architectures, one to invert the fully-connected (FC) layers and the other to invert the convolutional (Conv) and max-pooling (MP) layers of SVDNet. Here, "inverting a layer" is another way to refer to the inversion of the features that the layer generates. Similar to the FC8 feature inverter, the architecture of inversion models by Dosovitskiy and Brox [2016a] inspires the design of feature inverters for the other layers of the SVD model.

The majority of feature inverters need to upsample an input feature map. Similar to the FC8 feature inverter architecture, the other feature inverters perform upsampling and strided convolution in a single step by using up-convolutional layers that use 4×4 filters with 2×2 stride and 1×1 cropping and upsample an input feature map by a factor of 2. The number of Uconv layers depends on the dimensionality of the layer to be inverted. For example, the feature inverter to invert the 256-dimensional FC7 layer uses five Uconv layers (see Table 6.2), while the feature inverter to invert the Conv4 layer uses two Uconv layers (see Fig. 6.8). The feature inverters for the Conv1 and Conv2 layers in the SVD model do not use the Uconv layers, as for them the model generates features

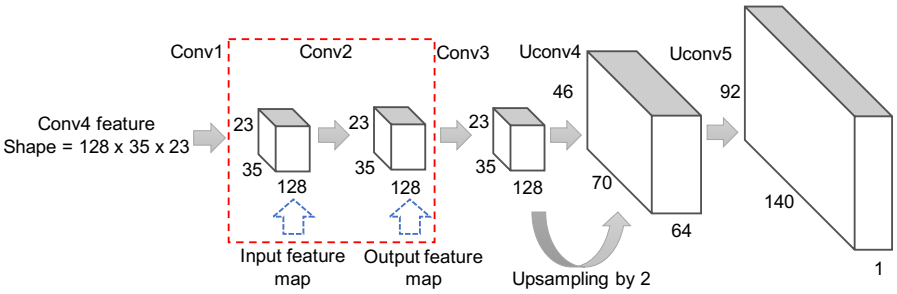

Figure 6.8: Feature inverter architecture for the Conv4 layer of the SVD model. The highlighted components refers to the 'Conv2' convolutional layer and its input and output feature maps. Conv and UConv refer to the convolutional and up-convolutional layers, respectively.

Layer	Input shape	N_{layers}	N_{conv}	N_{params}
FC8	64×1	11	4	$131,201$
FC7	256×1	10	4	$176,001$
MP6	$64 \times 11 \times 7$	5	1	$80,484$
Conv5	$64 \times 33 \times 21$	5	3	$144,772$
Conv4	$128 \times 35 \times 23$	5	3	$576,260$
MP3	$32 \times 37 \times 25$	6	4	$45,892$
Conv2	$32 \times 111 \times 76$	4	4	$28,324$
Conv1	$64 \times 113 \times 78$	2	2	$37,700$

Table 6.3: Overview of architectures of all the feature inverters in the SVD model. Layer: SVDNet layer a feature inverter inverts, Input shape: input to a feature inverter - ordered as: number of channels $\times$ time $\times$ frequency. N_{layers}, N_{conv} and N_{params} refer to the number of layers, the number of convolutional layers and the number of trainable parameters in a feature inverter, respectively. Conv: convolutional layer, FC: fully-connected layer and MP: max-pooling layer.

without using the max-pooling layers.

Similar to the FC8 feature inverter, the experiments increase the capacity of the other feature inverters by adding convolutional layers; either after every up-convolutional layer, except the last (for inverting an FC layer) or before the first up-convolutional layer (for inverting a Conv or MP layer). The number of convolutional layers used for inverting an FC layer is fixed to 4 as it depends on the number of UConv layers, but to decide the number of convolutional layers for inverting a Conv or MP layer, grid search was performed with the number of convolutional layers $\in \{1, 2, 3, 4, 5\}$. The convolutional layers perform strided convolution using 3×3 filters with 1×1 stride and 1×1 zero-padding and improve the visual appearance of the reconstructions [Dosovitskiy et al., 2017].

Similar to the FC8 feature inverter, all the layers use ELU non-linearity and batch normalisation layers. Moreover, except for the Conv1 and Conv2 layers, each feature inverter generates an inverted representation with a larger spatial

size and later trims it to match the input excerpt size (115×80). The feature inverters for the Conv1 and Conv2 layers generate an inverted representation of the same shape as input by symmetrically padding the missing dimensions.

The inverted representations need to have the same size as that of the inputs due to their use in computing the input space loss. Moreover, the inverted representations are also fed to the comparator model that was trained with excerpts of the same size as that of the inputs. There exist different ways to match the sizes of inputs and inverted representations. For example, by generating inverted representations of size smaller than inputs and then downsampling the inputs to the inverted representation size. However, the approach by Dosovitskiy and Brox [2016a] that generates inverted representations of size larger than the input size and later crops them to match the input size was used as it generated interpretable visualisations for image classification models.

Table 6.3 provides details about the total number of layers, the number of Conv layers and the number of learnable parameters in each feature inverter. Appendix A provides the detailed architecture of the feature inverters for the convolutional and max-pooling layers.

6.4.2 Training methodology

The experiments train the feature inverter for any SVDNet layer using features that the layer extracts from normalised log-scaled mel-spectrogram excerpts of around 1.6 seconds duration. The dimensions of features used to train a feature inverter depend on the layer to be inverted. For example, the experiments train the FC7 feature inverter using 256-dimensional features. Similarly, the experiments train the MP3 feature inverter using features of shape $32 \times 37 \times 25$. The experiments generate input mel-spectrogram excerpts from the Jamendo training dataset with a hop size of 10 frames. The rest of the training methodology remains the same as for training the FC8 feature inverter (See Section 6.3.4).

6.4.3 Quantitative evaluation of the feature inverters

The experiments train eight feature inverters using the Jamendo training dataset and the architectures and training methodology discussed earlier. The experiments evaluate the performance of each feature inverter on an evaluation set of 128 mel-spectrogram excerpts. The experiments build the evaluation set by randomly selecting 8 excerpts from each of the 16 audio files in the Jamendo test dataset. The experiments quantify the performance of feature inverters by calculating the average NRE for each feature inverter on the evaluation dataset.

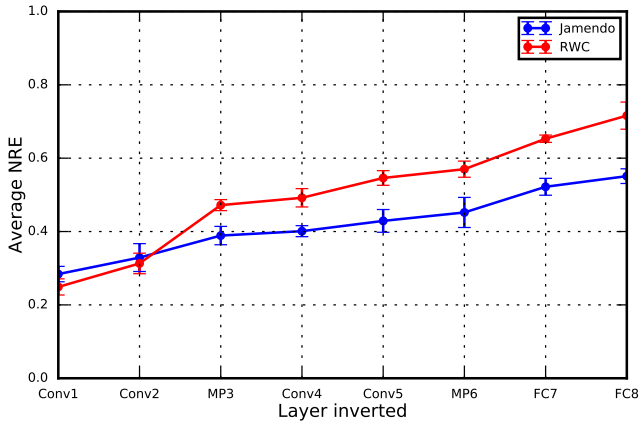

Figure 6.9: Performance evaluation of the feature inverters. The plot depicts the average normalised reconstruction error (NRE) for all the feature inverters of the SVDNet model. Layer inverted refers to the layer of the SVDNet model. The error bars represent standard deviation.

Mahendran and Vedaldi [2015] define NRE as

$$NRE = \frac{\|\mathbf{x}_i - \hat{\mathbf{x}}_{iL}\|_2}{N_c} \tag{6.6}$$

where N_c is a normalising constant computed from the average pairwise Euclidean distance between excerpts in the evaluation set.

The experiments also evaluate the feature inverters on the RWC dataset (see Chapter 3) to understand whether the results of the quantitative evaluation on Jamendo extend to the RWC dataset. The RWC dataset, unlike Jamendo, is not available pre-partitioned into separate subsets. Thus, to evaluate the feature inverters, the experiments first build an RWC test dataset by randomly selecting 20 audio files from a set of 100 and then using them to build an evaluation dataset of 160 randomly selected excerpts (8 excerpts per audio file). Moreover, in order to evaluate the feature inverters on a larger evaluation dataset, the experiments randomly sample 10 different evaluation sets from each dataset, calculate the average NRE for each and then take an average. Thus, effectively the experiments evaluate the feature inverters on evaluation datasets of size 1280 (Jamendo) and 1600 (RWC) excerpts.

Fig. 6.9 shows the results of the evaluation. For both the datasets, the reconstruction error is largest for the FC8 layer (the deepest layer in SVDNet) and consistently decreases for representations inverted from shallower layers. This is predictable as the dimensionality of the features in shallow layers is larger than in deep layers, assisting them to preserve more information. For instance,

if we compare the feature inverter for the deepest max-pooling layer (MP6) to the one for the first fully-connected layer (FC7), there is a large increase in the reconstruction error (by nearly 15% for Jamendo and 14% for RWC). This happens as the dimensionality reduction of features from MP6 to FC7 is about 19 times, compressing a 4928-dimensional feature to 256 dimensions. Similarly, the results depict a large increase in the average NRE between the feature inverters for the Conv2 and MP3 layers. This likely occurs due to max-pooling operation that compresses feature dimensionality by 9 times between the two layers. The results do not show such a significant change in reconstruction error between the Conv5 and MP6 layers, which seems due to the smaller feature map size at the Conv5 layer than at the Conv2 layer.

The results also depict that the feature inverters have larger reconstruction error on the RWC dataset at all but the first two layers. This is expected since both the discriminator (the SVD model) and the feature inverters are trained on the Jamendo dataset leading to some overfitting. One possible explanation for the comparable average NRE of the Conv1 and Conv2 layers is that these shallow layers of the model are learning low-level features that are generalisable across related datasets [Zeiler and Fergus, 2014]. This becomes less so at deeper layers, where features are likely tuned to specific traits of the training data.

6.4.4 Qualitative analysis of the inverted features

Fig. 6.10 shows visualisations from each layer of the SVD model. The experiments generate these visualisations by selecting four inputs, two from each test dataset (Jamendo and RWC). Out of the two inputs per dataset, one belongs to the vocal category and the other to the non-vocal category. The experiments use the feature inverters to invert the features that SVDNet layers extract from each input. The results provide some insights into the deep SVD model behaviour. For instance, reconstructions from the FC8 layer suggest that this layer does not retain the harmonic structures present in the inputs. Moreover, it appears that this layer preserves either the high frequency or the low frequency content of an input. Similarly, FC8 does not preserve any temporal information (musical onset locations) present in the inputs. Interestingly, for a large number of inputs (in addition to the four inputs in Fig. 6.10), the results depict that inverted representations (from this layer) for the vocal category inputs have energy in higher frequencies while inverted representations for the non-vocal category inputs have energy in lower frequencies.

Similarly, inverted representations from the FC7 layer suggest that this layer preserves some harmonic content and approximate onset locations of the inputs. However, there are some deviations from this behaviour. For instance, in Fig.

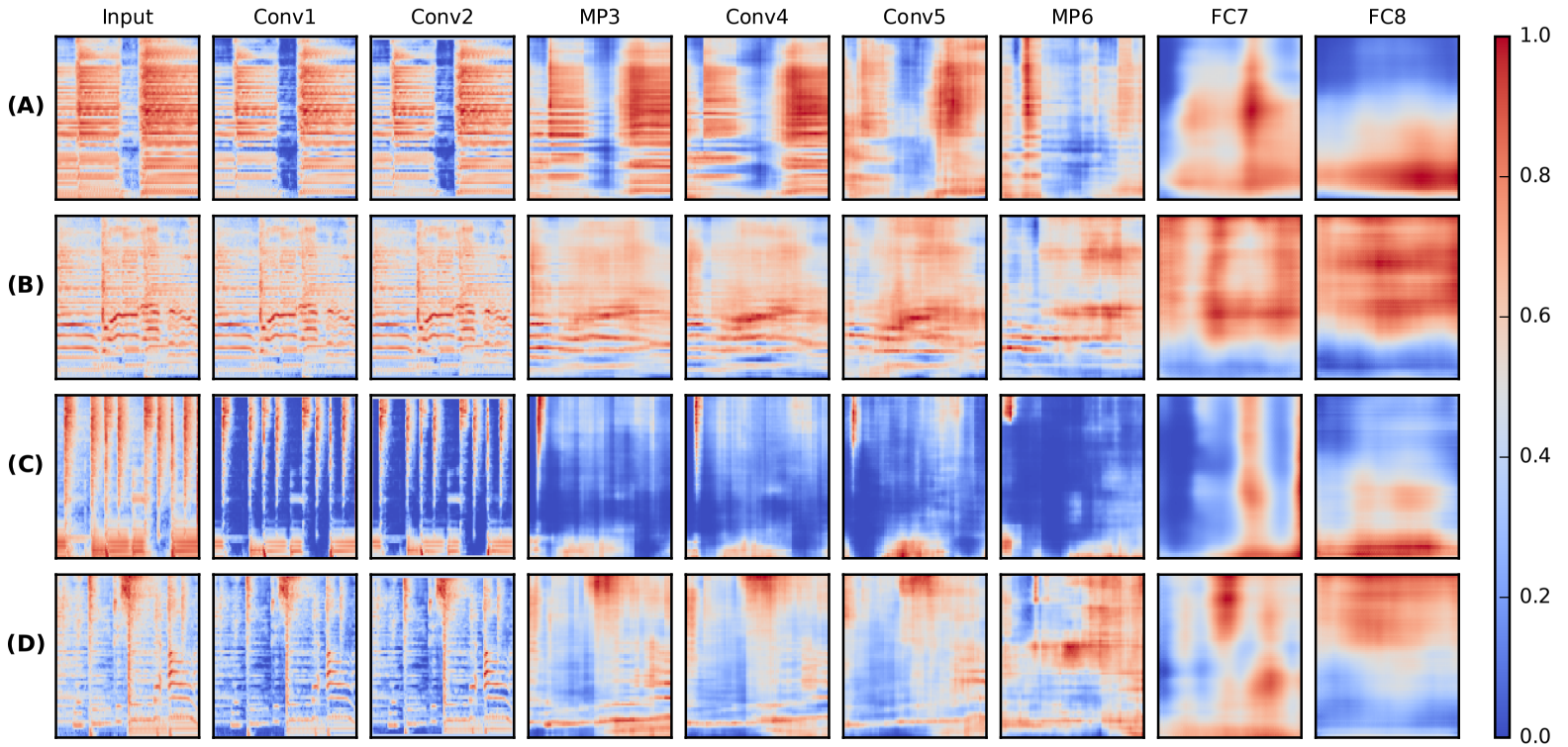

Figure 6.10: Feature inversion from successive layers of the deep SVD model. Each row corresponds to one input excerpt. (A), (B) are respectively non-vocal (time index : 1.43 seconds - 4.65 seconds) and vocal (time index : 33.0 seconds - 34.65 seconds) excerpts from "03 - Say me Good Bye.mp3" in the Jamendo test dataset. Similarly, (C) and (D) are respectively non-vocal (time index : 5.0 seconds - 6.65 seconds) and vocal (time index : 17.00 seconds - 18.65 seconds) excerpts from "RWC- MDB-P-2001-M04/5 Audio Track.aiff" in the RWC test dataset. Columns contain mel-spectrograms of (from left to right) the input signal and inverted representations from successive SVDNet layers (as labelled). The visualisations highlight how the model ignores aspects of the input content as it forms higher-level representations. Inversions from shallow layers resemble the input, but the reconstruction quality reduces for deeper layers. Conv, MP and FC refer to the convolutional, max-pooling and fully-connected layers, respectively.

6.10 row (B), the harmonic structures are less evident. Similarly, for the input in Fig. 6.10 row (C), the FC7 feature inverter is unable to reconstruct all the harmonic and temporal content present in the input. This seems due to the reason that the experiments do not train the feature inverters on RWC, thus the reconstruction error is higher for this input, resulting in poor reconstruction.

The results depict that inverted representations from the deepest convolutional layer of the SVD model (Conv5) contain more information than those from the two fully-connected layers. For both the inputs from the Jamendo dataset (Fig. 6.10 rows (A) and (B)), the model preserves much of the input content (e.g., the reconstructions capture the harmonic structure and approximately align the temporal boundaries with the input). This confirms results from the quantitative evaluation of Conv5 and FC7 feature inverters that showed that the average NRE for the Conv5 layer feature inverter is about 18% less than for the FC7 feature inverter. The visualisations for the RWC excerpts (Fig. 6.10 rows (C) and (D)) depict similar results. Finally, reconstructions from all the other layers follow a similar pattern - moving toward shallower layers, they become visually similar to the input, increasingly showing the presence of finer harmonic and temporal structures. Moreover, the inversions from Conv1 and Conv2 are very close to the respective inputs. This suggests that the filters of the first two convolutional layers act as a bijective map, e.g., performing an invertible frequency transform. Moreover, the visualisations from deeper layers in the model are more blurry than from shallow layers. This suggests that deeper layers capture more invariances from data than shallow layers.

6.4.5 Analysing FC8 features

The visualisations from Section 6.4.4 showed that inverted representations from the FC8 layer corresponding to the vocal and non-vocal category inputs contain energy in the higher and lower frequencies, respectively. This section describes an experiment that further analyses this observation. To do this, the experiment uses a 20 songs subset from the MedleyDB dataset (see Section 3.2). Each song in the subset contains vocal and non-vocal stems and their mix. The experiment randomly selects two songs from the subset. For each song, in addition to the available mix of vocal and non-vocal stems, the experiment creates five more mixes by first attenuating the vocal stem in multiples of 6 dB and then by mixing the attenuated vocal stems with the non-vocal stem. This section calls the available mix 'Mix^{avail}' and the synthesized mixes 'Mix_k^{synth}' where k $\in \{1, 2, 3, 4, 5\}$. Thus, for instance, Mix_3^{synth} represents an audio file where the experiment mixes a non-vocal stem with an 18 dB attenuated vocal stem.

For each of the two songs, the experiment analyses FC8 features from the

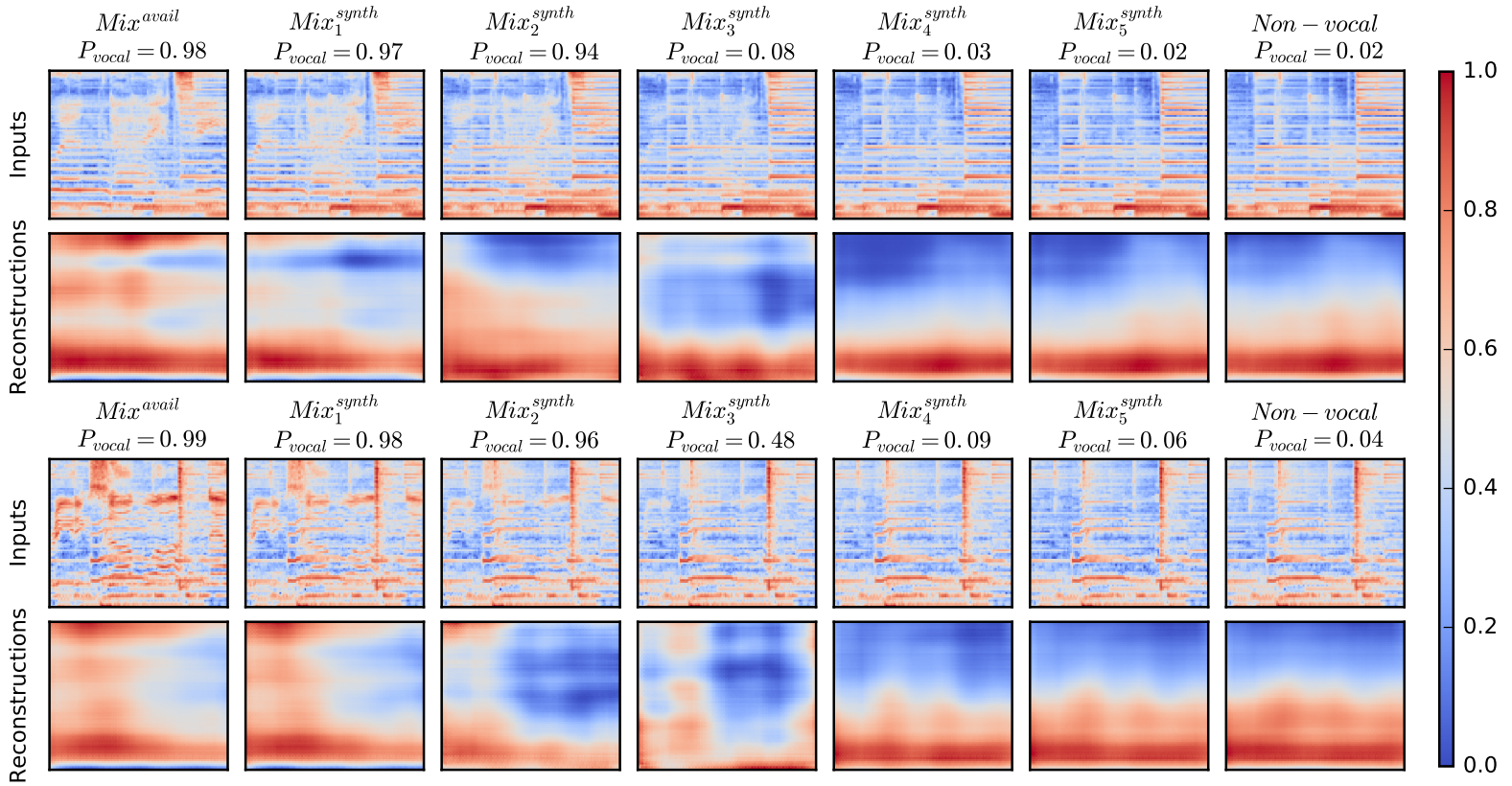

Figure 6.11: Inversion of FC8 features from inputs with varying strength of vocals. The first and last two rows depict visualisations for inputs from the "LizNelson_Rainfall", "AlexanderRoss_VelvetCurtain" songs in the MedleyDB dataset, respectively. For each song, each column depicts an input excerpt and its reconstruction using its FC8 features. For each song, Mix^{avail}, Mix_k^{synth} and Non-vocal represent an input excerpt extracted (at the same time index) from the available mix, synthesized mixes and the non-vocal stem, respectively. P_{vocal} represents the model's confidence that an input excerpt contains vocals. The experiment extracts excerpts for the first and second song at time offsets 15 seconds and 115 seconds, respectively.

seven audio files (the six mixes and the non-vocal stem). To do that, the experiment first randomly selects two time indices, one for each of the two songs. Then, for each of the song, the experiment samples the mel-spectrogram excerpt corresponding to the selected time index from each of the seven audio files. Finally, the experiment uses the feature inversion method to analyse the input information that the FC8 layer preserves for each of the seven audio files per song. The experiment aims to visualise how the information in the inverted representations varies as the energy of vocal components reduces progressively.

Fig. 6.11 depicts the results of the experiment. The first two and the last two rows depict inputs and their reconstructions using the FC8 features from the first and second song, respectively. Each column depicts an input, the model's confidence score that this input contains vocals P_{vocal}, and the reconstruction of the input from the FC8 features. The first column in each song corresponds to visualisations for the excerpt sampled from 'Mix^{avail}' audio file and on moving from left to right the energy of the vocal component in the excerpt reduces by 6 dB. The last column in each song corresponds to visualisations from the non-vocal audio file.

The visualisations from both the songs depict that when the model predicts an input contains vocals with high confidence ($P_{vocal} \geq 0.9$), all the reconstructions contain some information in the high frequency regions (frequency ≥ 4096 Hz). On the other hand, such an information is mostly absent in cases when the model predicts an input contains non-vocals. For the song 2, the reconstruction from the Mix_3^{synth} contains some information in high frequency regions, but the prediction probability suggests that the model is poorly confident about its prediction. In fact, when the model predicts an input to contain non-vocals with high confidence ($P_{vocal} \leq 0.1$) the reconstructions have information only in the low frequency regions (frequency ≤ 1024 Hz). Moreover, for each song, the visualisations depict that as the energy of the vocal component reduces, the reconstructions change from containing information both in the low and high frequency regions to only in the low frequency regions. These results further support the observations from Section 6.4.4 that inverted representations from the FC8 layer contain energy in the higher and lower frequencies for the vocal and non-vocal categories' inputs, respectively.

6.5 Summary and conclusion

This chapter presented feature inversion - an interpretable machine learning method that maps DNN features back to the input space. The chapter discussed an approach of feature inversion that first trains an up-convolutional neural network (feature inverter) by minimising the input and feature space losses

and then uses the feature inverter to invert a DNN feature. The chapter used the feature inversion method in two ways - to explain DNN predictions and to analyse DNN features.

The first experiment proposed a novel method that uses feature inversion to explain predictions of a DNN. The method inverts a feature from the deepest layer of a DNN and uses the inverted representation to generate a binary mask to mask the input. The unmasked portion of the input highlights the region influential to a prediction. The chapter demonstrated the proposed method for a state-of-the-art deep singing voice detection model. The chapter first discussed the architecture and training methodology of the deepest layer feature inverter and then demonstrated the explanation generation process for a randomly selected instance. The chapter used the explanations for some carefully selected instances to analyse the trustworthiness of the model. In particular, the chapter generated explanations for instances having vocal and non-vocal components in separate temporal segments to analyse if the model is predicting by extracting information from the appropriate segments. Finally, the chapter evaluated the loss associated with the proposed explanation method by generating SVDNet predictions for the masked versions of inputs. The evaluation results for two publicly available datasets suggested that explanations from the proposed method highlighted influential regions in about 80% (masking threshold = 0.5) of the randomly chosen inputs.

The results from the first experiment clearly suggest that inverting features from the deepest layer assists in highlighting the influential regions in inputs. Although there is some explanation loss, it is reasonable to believe that an accurate feature inverter and carefully chosen masking threshold will help in improving the performance of the explanation method. Although there are other popular approaches for explaining DNN predictions (see Chapter 4), either they are computationally expensive or use auxiliary information or generate finer explanations which may not be useful for understanding machine listening models. The proposed explanation method is computationally expensive in training the feature inverter, but the explanation generation is very fast as it requires just a forward pass through the feature inverter.

The second experiment applied the feature inversion method to understand a machine listening model (SVDNet). Feature inversion helped to understand the global behaviour of the model by visualising the information preserved by any layer in the model. The experiment trained eight up-convolutional neural networks - each to invert features from one of the layers of SVDNet. The experiment quantitatively analysed the feature inverters for each layer in the model to understand the change in input reconstruction error across different layers. The results demonstrated that the average NRE changes by about 15%

for the Jamendo and RWC datasets between the MP6 and FC7 layers due to high dimensionality reduction. Moreover, the experiment qualitatively visualised the inverted representations to understand the input content preserved by any layer in the model. The results suggested that the deepest fully-connected layer does not retain any of the temporal or harmonic structures present in an input. The experiment further analysed FC8 reconstructions for mixes with varying levels of vocal content. The results indicated that when the model confidently predicted the presence of vocals in an input, the FC8 inversions have information in the high frequency region, a phenomenon not seen for the cases where the model confidently predicts the absence of vocals. One possible explanation for this behaviour is the presence of fricatives in singing voice that result in information in the high frequency regions (frequency $\geq$ 4096 Hz). Qualitative analysis of other layers revealed that the FC7 layer preserves some harmonic and temporal information of an input while the reconstructions from the other layers are visually similar to the input.

Thus, the results of the experiments in this chapter suggest that feature inversion is a versatile method that can explain a DNN prediction as well as analyse its latent features.

6.6 Reproducibility

The code for all the experiments is open-sourced. The below mentioned Github repositories contain the code and parameters to reproduce the thesis results and steps to generate new results.

1. Experiment 1 (explaining SVDNet predictions) - `https://github.com/saum25/EUSIPCO-2018`
2. Experiment 2 (understanding SVDNet features) - `https://github.com/saum25/ISMIR-2018`

Chapter 7

Conclusions and future work

This thesis presented three methods to analyse the behaviour of machine listening models. The first method (SLIME) analyses the models locally, the second method (activation maximisation) analyses the models globally, and the third method (feature inversion) analyses the models both globally and locally. The thesis demonstrated the methods for five SVD models that aim to detect the presence of vocals in musical excerpts. This chapter summarises the key results from the experiments and highlights the contributions this thesis makes to interpretable machine learning and machine listening research (Section 7.1). Moreover, this chapter presents some ideas to extend this research (Section 7.2) and discusses potential research directions for interpretable machine learning (Section 7.3).

7.1 Summary

Chapter 2

Chapter 2 provided a survey of IML that involves designing and applying methods to understand the behaviour of machine learning models. The chapter defined what interpretability means in the context of this thesis and discussed scenarios where interpretability is essential (e.g., in tackling the problem of incomplete model specifications). The chapter described two main categories of IML methods: methods to design inherently interpretable models and methods for post-hoc interpretability, and discussed several methods within each category. Finally, the chapter discussed some recent work on analysing machine listening models. The survey provides an understanding of the scope (local or global), strengths, and weaknesses of different IML methods. Moreover, it informs about the existing approaches to analyse machine listening models and

their limitations. This understanding was helpful to identify methods suitable to answer the research questions outlined in Chapter 1.

Chapter 2 also introduced SVD, the machine listening use case this thesis used for experiments. The chapter defined the use case, mentioned some of its applications, discussed different approaches (e.g., feature engineering) to design SVD models, and highlighted the state-of-the-art methods within each category. This thesis selected SVD as it is a well-established MIR task with several proposed approaches and benchmarked publicly available datasets, the state-of-the-art model is a fairly sophisticated open-sourced model that reports very accurate performance on the benchmarked datasets [Schlüter and Grill, 2015], and the classification categories (vocal and non-vocal) are highly interpretable and distinct.

Chapter 3

Chapter 3 introduced four datasets and five SVD models that this thesis used for the experiments in Chapters 4–6. The thesis used Jamendo for both training and analysing the SVD models and the other three datasets (RWC, MedleyDB, and ccMixter) only for analysing the SVD models. The chapter also described the five SVD models: two shallow models (a BDT and an RF model) and three deep models. The shallow models used the median and standard deviation of the first 30 MFCCs and their first-order derivatives over 10 audio frames as features. The chapter described the training methodology of the shallow models and reported their performance on Jamendo. The classification accuracies of the RF, BDT, and a baseline model (that classified all inputs as vocals) on the test dataset are $76.3\%, 71.4\%$, and 57.5%, respectively, suggesting that the shallow models have learnt some representation of vocals.

Chapter 3 also discussed the state-of-the-art deep SVD model (SVDNet) that assigns scores to mel-spectrogram excerpts of around 1.6 seconds, indicating its confidence about the presence of vocals in input excerpts. The chapter described the architecture, training methodology, and performance of the model on the Jamendo test dataset. The chapter also reported the performance of two more deep SVD models used in this thesis. The first model (SVDNet-R1) is a replica of SVDNet but ported and re-trained using Tensorflow. The second model (SVDNet-R2) was also ported and re-trained using Tensorflow but after replacing the sigmoidal output neuron in SVDNet-R1 by two fully connected neurons followed by a softmax layer. SVDNet-R1 and SVDNet-R2 reported performance similar to SVDNet on the Jamendo test dataset.

Chapter 4

Chapter 4 introduced SLIME, a method to generate interpretable explanations for predictions of shallow and deep machine listening models. SLIME extends the LIME algorithm [Ribeiro et al., 2016b] to machine listening by defining temporal, spectral, and time-frequency interpretable representations. These representations generate explanations highlighting the most influential temporal, spectral, and time-frequency components in an input. The chapter reported two experiments that demonstrated the effectiveness of SLIME for validating the trustworthiness of three SVD models. The first experiment analysed the temporal explanations for four instances from the Jamendo test dataset to identify that the BDT model is untrustworthy as it classified two instances to the vocal category with high confidence by using information from the non-vocal components. The second experiment analysed the positive and negative time-frequency explanations for an instance with separate vocal and non-vocal temporal segments to validate the behaviour of SVDNet for that instance. The chapter also quantitatively compared time-frequency explanations from SLIME with saliency maps (SMs) [Zeiler and Fergus, 2014] to analyse whether SLIME explanations accurately identified influential input components. The results suggested that there was a fair numerical agreement between explanations from the two methods.

Chapter 4 also described two further experiments that aimed to analyse the robustness of SLIME to changes in two input parameters: the number of perturbed samples N_s and the content of synthetic components. The first experiment demonstrated that SLIME explanations are sensitive to N_s and SLIME needed high N_s (e.g., $N_s \geq 50000$) to generate stable explanations. However, a large N_s increased the explanation generation time and thus, the selection of a suitable N_s should consider both the stability of explanations and the explanation generation latency. The second experiment proposed some novel content types for input perturbation in SLIME and used them to demonstrate that explanations from SLIME are sensitive to the content of synthetic components. This is an important observation as such behaviour is highly likely to extend to explanation methods that perturb inputs by occlusion [Zeiler and Fergus, 2014, Ribeiro et al., 2016b]. Finally, the chapter introduced a novel method to generate reliable explanations from SLIME and demonstrated it for a dataset synthesised using vocal and non-vocal stems. The method uses the ground-truth explanations to identify suitable content types for SLIME. The results suggested that for the temporal explanations corresponding to 656 instances, among four content types, the average bin magnitude of an input seemed to be the most suitable content type as it generated explanations with at least two super-samples common with the ground-truth explanations for around 84% of

the instances.

Chapter 5

Chapter 5 presented AM, a method that globally analyses deep machine learning models. AM synthesises examples in the input space by iteratively optimising noise vectors to maximally (or minimally) activate DNN components (e.g., neurons). Chapter 5 presented a novel method for AM that uses a GAN as a regulariser in the AM pipeline to ensure that the synthesised examples are realistic. The proposed method allows to analyse DNN components without retraining the GAN if the GAN output domain matches the DNN input domain. The chapter also introduced a novel method that uses the FID metric [Heusel et al., 2017] to select suitable hyper-parameter configurations for AM. The method makes the hyper-parameter selection process scalable and assists in effectively searching the hyper-parameter space for any DNN component. Both proposed methods are domain-independent and thus applicable to pre-trained DNNs from any domain (e.g., images, audio, text). Importantly, the interpretability of synthetic examples from the proposed methods also depends on the performance of the GAN prior. However, training a suitable GAN might be challenging for certain use cases.

Chapter 5 also described two experiments that demonstrated the proposed methods for the SVDNet-R1 and SVDNet-R2 models. The first experiment synthesised mel-spectrograms that maximally and minimally activated the output neuron in the SVDNet-R1 model. The results suggested that the configuration selection method successfully ranked configurations according to the interpretability of the set of examples synthesised by them. Moreover, the method effectively identified configurations that can potentially synthesise adversarial examples. The experiment further used the best configurations to synthesise examples that maximised and minimised the pre-sigmoid activation of the output neuron. The analysis of the synthesised examples demonstrated that the neuron captured high-level class concepts, i.e., the examples that maximally and minimally activated its activation contained the presence of interpretable vocal and non-vocal characteristics. The second experiment used the proposed methods to synthesise examples that maximally activated each output layer neuron in the SVDNet-R2 model. The results demonstrated that both the methods effectively generalised to SVDNet-R2. Moreover, the examples synthesised using the best configurations demonstrated that both neurons captured high-level class concepts. Finally, the chapter discussed whether examples synthesised for SVDNet-R2 were qualitatively more interpretable than those for SVDNet-R1. The results suggested that although some examples for SVDNet-R2 were

comparatively less noisy and resulted in higher activation scores, overall, the examples synthesised for both the models were perceptually quite similar, suggesting that the single neuron can effectively learn interpretable representations despite the influence of inter-class complexities.

The chapter also presented a user study that aimed to further analyse some observations from the qualitative analysis of the two models. The study was done online, and it consisted of two listening tests that required participants to listen to several audio examples and answer the corresponding questions. The analysis of responses from 23 participants further supports two observations - maximally and minimally activating examples corresponding to SVDNet-R1 contain vocal and non-vocal characteristics, respectively and the examples synthesised for the output neuron in SVDNet-R1 are at least as intelligible as the corresponding examples for the output neurons in SVDNet-R2. The study also analysed the intelligibility level of the synthesised examples. The analysis of responses suggests that the participants found the examples fairly unintelligible, however, due to poor agreement among the participants, further analysis would be required to better understand the responses for this question.

Chapter 6

Chapter 6 discussed a method that can both locally and globally analyse deep machine listening models. Specifically, the chapter presented feature inversion, a method to map DNN features back to the input space to understand the information a DNN preserves in each of its layers. The chapter proposed to perform feature inversion by training up-convolutional neural networks through the minimisation of the feature space and input space losses [Dosovitskiy and Brox, 2016b]. Recent works demonstrated feature inversion for the global analysis of deep image classification models [Mahendran and Vedaldi, 2015, Dosovitskiy and Brox, 2016a]. This chapter demonstrated that feature inversion is a versatile method that can analyse DNNs both locally and globally.

This chapter introduced a novel domain-independent method for explaining DNN predictions. The method used feature inversion to invert the deepest hidden layer feature of a DNN and convert the inverted representation to a binary mask that is applied to an input. The unmasked input sections highlight the regions influencing a prediction. The chapter qualitatively validated the method by explaining SVDNet predictions for two instances with separate vocal and non-vocal temporal sections. Moreover, the chapter quantitatively verified whether regions highlighted by the method influence model predictions by feeding the explanations back to the DNN and analysing the change in prediction label. The results for SVDNet for random instances from Jamendo and

RWC suggested that for around 80% of the instances, SVDNet predictions do not change, demonstrating that feature inversion is an effective method.

Chapter 6 also used feature inversion to quantitatively and qualitatively analyse the input information preserved by each layer in the SVDNet model. The quantitative analysis computed the average NRE for each layer using random instances from the Jamendo and RWC datasets and demonstrated that the first and deepest layers of the model preserved the maximum and minimum information from an input. The qualitative analysis demonstrated that the deepest layer in the model did not preserve any temporal or spectral information from an input. Moreover, the inverted representations from this layer corresponding to the vocal and non-vocal excerpts contain energy mostly in the higher and lower frequency regions, respectively. The experiments further analysed this observation by qualitatively examining the change in inverted representations with uniform attenuation of the vocal content in a vocal and non-vocal mix. The visualisations from the other layers suggested that the FC7 layer preserved some temporal and harmonic information from the input and the reconstructions from the other layers were visually similar to the input.

7.2 Potential research directions

This section presents some novel ideas to extend the research in this thesis. Some of these ideas aim to address the limitations of the presented research, while the others aim to further validate results from this research by either performing large-scale experiments or by using alternate approaches.

Chapter 4

- Section 4.4.2 demonstrated that SLIME explanations are sensitive to the content of the synthetic components. It would be intriguing to examine if this behaviour extends to other methods that perturb inputs by occlusion [Zeiler and Fergus, 2014]. Moreover, it would be important to validate the result from Section 4.4.2 for other models and input data domains to ensure that they do not influence the results.

- SLIME explanations highlight N input components, where N is pre-defined and generally small to limit the complexity of explanations. However, explanations generated using this approach may miss some influential components (if the number of such components is greater than N) and may include less influential components (if the number of influential components is less than N). Thus, it may be useful to devise a method that

can automatically select influential components (e.g., using the weights SLIME assigns to all input components).

- The experiments in Chapter 4 segmented inputs at pre-defined fixed boundaries. This helps to provide some insights into model behaviour, but it may be more interpretable if SLIME performs segmentation at more meaningful boundaries (e.g., temporal onset locations, frequency bins corresponding to harmonic components [Müller, 2015]). Thus, an important future work could be to propose new interpretable representations for SLIME.

- Another future work can be to design methods for examining the behaviour of SLIME. This would assist in highlighting scenarios where SLIME depicts inconsistent or incorrect behaviour. For example, one can analyse whether SLIME explanations remain the same if inputs are modified using label-preserving transformations (e.g., increasing input loudness).

- Section 4.4.3 presented an approach to select the most suitable content type for SLIME. However, this approach does not work in the absence of ground-truth annotations. Thus, another future research topic could be to design more methods for generating reliable predictions from SLIME.

- Section 4.4 analysed SLIME behaviour for the deep SVD models. It would be interesting to analyse whether the observation that SLIME explanations are sensitive to the two input parameters extends to the shallow SVD models.

- Another future work could be to analyse the behaviour of SLIME for other input parameters (e.g., weight function, distance measure).

Chapter 5

- The experiments in Chapter 5 analysed the concepts captured in the output layer neurons. It would be an interesting future work to use the proposed methods to analyse the concepts captured in other DNN components (e.g., neurons in convolutional layers). It would help to analyse whether deep machine listening models capture features hierarchically.

- The analysis of the responses for the intelligibility level of the synthesised examples shows that the participants found the examples fairly unintelligible, but with a poor agreement. It would be interesting to further analyse the responses to understand the reasons behind the poor agreement. For example, do the responses differ due to differences in understanding of intelligibility?

- The use of priors (hand-crafted or learned) in the AM pipeline aims to generate realistic examples. However, it remains unclear how much the priors influence the content in synthesised examples. Thus, it is intriguing to analyse whether a strong prior (e.g., a GAN) can highly influence the content in synthesised examples to such an extent that the examples do not accurately reflect the features captured by the corresponding DNN components.

Chapter 6

- The feature inversion method in Chapter 6 uses two loss functions each computing the squared Euclidean distance in the input space (spectral domain) and the feature space, respectively. However, some interesting questions remain. For example, it is not evident how inverted representations will change on replacing the Euclidean distance by the L_1 norm that encourages sparsity. The L_1 norm has been used recently for neural audio synthesis [Défossez et al., 2018, Engel et al., 2020]. Moreover, although the spectral domain loss in Chapter 6 can be considered as a variant of the perceptual loss by Défossez et al. [2018], it would be interesting to investigate whether the use of sophisticated spectral losses (e.g., multi-scale spectral loss [Engel et al., 2020]) improves the perceptual quality of the inverted representations.

- This chapter used a feature inversion method that trains up-convolutional networks to invert features. This learning of deterministic functions generates high-quality visualisations but provides limited insights into the invariances captured by DNN layers. Thus, potential future research could be to use the approach of Mahendran and Vedaldi [2015] to understand the invariances captured by SVDNet layers (e.g, pitch invariance).

- It is important to note that the up-convolutional method implicitly learns a prior, but the above mentioned method requires priors in the optimisation objective. Thus, another future work related to the first one could be to design hand-crafted priors for the audio domain. We can use the existing priors, but they are either generic (e.g., α-norm) or specifically designed for images (e.g., total variation norm [Mahendran and Vedaldi, 2015]). Hand-crafted priors will also be useful for AM.

- Another future work could be to compare the inverted representations from the two feature inversion methods to validate if the conclusions in Section 6.4.4 for the SVDNet model are consistent. This could be further analysed quantitatively by computing the amount of energy in different

frequency bands to understand if the inverted representations for inputs from the two categories differ in the amount of energy in different frequency bands.

- The explanation method uses a binary mask created from an inverted representation by using a simple normalisation and thresholding procedure. It could be a useful experiment to modify the way the explanation method uses an inverted representation and analyse whether the new approach improves the performance (lower explanation loss for the best threshold) of the explanation method. For example, one can perform normalisation and thresholding by using reconstruction error values instead of using reconstructed features themselves. Similarly, one can first multiply the inverted representation with the input and then normalise and threshold the output.

Finally, from a more general machine listening perspective, it would be interesting to use the proposed methods to analyse models trained for other machine listening use cases (e.g. onset detection, F0 estimation [Müller, 2015], environmental sound classification [Piczak, 2015]). Moreover, it would be intriguing to understand the potential of the proposed methods in multi-label classification scenarios as opposed to single-label classification tasks.

7.3 Discussion on interpretable machine learning

This section presents a general discussion on some of the key challenges in IML research and highlights how this thesis contributes to addressing some of those challenges. Thus, this discussion aims to provide further insights into potential directions in IML research.

Lack of formalism

IML research suffers from a lack of consensus about the definitions of key terms (See Section 2.2.1). For example, there is a lack of clarity about what "interpretability", "explainability", and "transparency" refer to in the context of machine learning. Researchers have been vocal about the lack of formalism in IML research [Doshi-Velez and Kim, 2017, Lipton, 2016], and some recent works have tried to address this issue by defining an IML vocabulary and introducing taxonomies to categorise IML algorithms [Rudin, 2019, Gilpin et al., 2018]. However, the lack of consensus still prevails. To prevent ambiguity, Section 2.2.1 defined the three terms (interpretability, interpretable models, and IML) used in this thesis.

Interpretable models vs post-hoc interpretability

Chapter 2 presented two main ways for ML interpretability: training inherently interpretable models and generating post-hoc explanations for pre-trained models. Recent years have witnessed a growing discussion about identifying the better of the two interpretability approaches. There are diverging opinions about the two approaches. Rudin [2019] argues that training interpretable models is a better approach, and we must not use post-hoc explanations for safety critical applications. Similarly, Ribeiro et al. [2016a] suggest to use interpretable models even for non safety critical applications, if these models can achieve the desired performance and use a restricted number of model components (e.g., features in a linear model).

However, some works argue in favour of post-hoc explanations, as training models with an interpretability constraint often restricts their predictive capacity [Ribeiro et al., 2016a,b]. Thus, the state-of-the-art models for complex real-world applications (e.g., speech recognition, image classification) are black-box models (e.g., DNNs). Moreover, using post-hoc explanations is the only approach if we re-use existing sophisticated models that are highly expensive to train (e.g., VGGNet). However, recent research has shown that post-hoc explanations may be unreliable [Kindermans et al., 2017]. Additionally, Chen et al. [2019] demonstrated using a complex training procedure that interpretable models can perform on par with state-of-the-art models for complex tasks.

Thus, it seems that both approaches have pros and cons, and this makes approach selection a context-dependent and subjective task. Moreover, further research is required for training high-performing interpretable models and improving the reliability of post-hoc explanations. This thesis used the post-hoc interpretability approach as one of the research objectives involved analysing the performance of the state-of-the-art SVDNet model.

Global analysis vs local explanations

Chapter 2 discussed approaches for post-hoc interpretability that analyse the global or the local behaviour of a model. The selection of a post-hoc analysis approach generally involves multiple factors (e.g., the motivation of analysis, the end-user of an explanation). Overall, explaining model predictions is comparatively simpler and quicker than understanding a model globally. However, local explanations may be unreliable and inconsistent. On the other hand, the global analysis is complex and usually time-consuming but provides a detailed and consistent analysis of model behaviour. This thesis used both approaches for post-hoc interpretability as the thesis aimed to both verify model trustworthiness and understand how a model constructs its predictions.

Timing information

The time taken to generate and analyse post-hoc explanations is an important metric for the design and selection of analysis methods. Explanation generation from some methods can be prohibitively slow [Zintgraf et al., 2017] and some methods may generate complex explanations (e.g., including several rules) increasing their interpretation time. Despite their importance, research rarely reports the timing information for the IML methods. This thesis analyses the timing metric for SLIME and suggests to use the metric as one of the criteria in selecting a suitable N_s. The other two methods (AM, feature inversion) are expensive offline but are near real-time online.

Reliability of local explanations

Recent research has demonstrated that some local explanation methods may generate unreliable and inconsistent explanations [Kindermans et al., 2017]. Similar results have motivated researchers to propose properties (e.g., completeness, implementation invariance, sensitivity [Sundararajan et al., 2017], and input invariance [Kindermans et al., 2017]) that should be inherent to every explanation method. Moreover, some works have stressed the need for quantitative evaluation frameworks to compare the performance of different explanation methods [Montavon et al., 2018]. However, researchers proposed the above properties and evaluation metrics for gradient-based attribution methods, and it is not evident if they are suitable for methods that explain predictions in terms of input regions (e.g., LIME, Occlusion). Alternatively, one can design novel metrics reflecting the reliability of explanations for these methods. For example, SLIME can provide a confidence score reflecting how well it approximated the non-linear decision boundary by an interpretable model.

This thesis quantitatively evaluated some methods. Chapter 6 performed a quantitative evaluation to demonstrate the reliability of the proposed local explanation method. Chapter 4 quantitatively demonstrated that SLIME may generate unreliable explanations for some content types.

More research questions

In addition to the earlier discussion, there are many more interesting research questions in IML. Some of the questions are mentioned below.

- Do IML methods designed and demonstrated for one data domain (e.g., image) generalise to other domains (e.g., audio)? This research faced this question and it seems that extending methods to other domains is challenging at two steps: in setting up the explanation generation pipeline, and

in interpreting the generated explanations. For example, SLIME proposed new interpretable representations to extend LIME. Similarly, to interpret AM synthesis results, the experiments involved auralisation.

- How can one expand current IML methods or develop novel IML methods for effectively analysing sequential models (e.g., recurrent neural networks)? There exist some methods that aim to analyse these models [Lei et al., 2016, Li et al., 2016, Karpathy et al., 2015], but their number and scope are very limited in comparison to existing approaches for analysing CNNs.

- Explanation methods that perform sensitivity analysis by input perturbation (e.g., by occlusion) use model predictions for instances that (due to input perturbation) are very different from the training data. An ML model may behave in an undefined manner for such perturbed samples. Thus, how can one examine the reliability of such explanations methods?

To summarise, this thesis presented and demonstrated three different post-hoc analysis methods to investigate the behaviour of machine listening models. Moreover, some of the presented methods are generic and thus assist in analysing the behaviour of ML models trained for other domains. Hopefully, this work would help develop robust and generalisable ML models and motivate further research in the analysis of machine listening models.

Appendix A

Feature inverter architectures

Chapter 6 discussed experiments that used feature inversion to analyse the features that each layer of the deep SVD model extracts from any input. The experiments trained eight feature inverters, one for inverting features from each layer of SVDNet. The chapter provided an overview of the key architectural components of feature inverters and discussed in detail the architectures of the FC8 and FC7 feature inverters. This appendix describes in detail the architectures of feature inverters for inverting the convolutional (Conv1, Conv2, Conv4 and Conv5) and max-pooling (MP6, MP3) layers of SVDNet.

In each of the feature inverter architectures described below, Conv, UConv and MP refer to the convolutional, up-convolutional and max-pooling layers, respectively. As discussed in Section 6.4.1, UConv layers perform unpooling (by a factor of 2) and strided convolution with 1×1 cropping, Conv layers perform strided convolution with 1×1 zero-padding to keep an output feature map of the same shape as the input. Moreover, for each feature inverter, the input and output shapes are ordered as: number of channels $\times$ time $\times$ frequency. During training both the input and output shapes get appended by the batch size dimension that the tables below do not show. $N_{filters}$ refers to the number of Conv or UConv layer filters. The Slice layer crops an upsampled feature map to the input excerpt shape, and the pad layer symmetrically pads each axis by N_{zeros}.

Layer	Input shape	$N_{filters}$	Filter	Stride	N_{zeros}	Output shape
Conv1	$64 \times 113 \times 78$	64	3×3	1×1	-	$64 \times 113 \times 78$
Conv2	$64 \times 113 \times 78$	1	3×3	1×1	-	$1 \times 113 \times 78$
Pad	$1 \times 113 \times 78$	-	-	-	1	$1 \times 115 \times 80$

Table A.1: The architecture of the Conv1 feature inverter.

Layer	Input shape	$N_{filters}$	Filter	Stride	N_{zeros}	Output shape
Conv1	$32 \times 111 \times 76$	32	3×3	1×1	-	$32 \times 111 \times 76$
Conv2	$32 \times 111 \times 76$	32	3×3	1×1	-	$32 \times 111 \times 76$
Conv3	$32 \times 111 \times 76$	32	3×3	1×1	-	$32 \times 111 \times 76$
Conv4	$32 \times 111 \times 76$	1	3×3	1×1	-	$1 \times 111 \times 76$
Pad	$1 \times 111 \times 76$	-	-	-	2	$1 \times 115 \times 80$

Table A.2: The architecture of the Conv2 feature inverter.

Layer	Input shape	$N_{filters}$	Filter	Stride	Output shape
Conv1	$32 \times 37 \times 25$	32	3×3	1×1	$32 \times 37 \times 25$
Conv2	$32 \times 37 \times 25$	32	3×3	1×1	$32 \times 37 \times 25$
Conv3	$32 \times 37 \times 25$	32	3×3	1×1	$32 \times 37 \times 25$
Conv4	$32 \times 37 \times 25$	32	3×3	1×1	$32 \times 37 \times 25$
Uconv5	$32 \times 37 \times 25$	16	4×4	2×2	$16 \times 74 \times 50$
Uconv6	$16 \times 74 \times 50$	1	4×4	2×2	$1 \times 148 \times 100$
Slice	$1 \times 148 \times 100$	-	-	-	$1 \times 115 \times 80$

Table A.3: The architecture of the MP3 feature inverter.

Layer	Input shape	$N_{filters}$	Filter	Stride	Output shape
Conv1	$128 \times 35 \times 23$	128	3×3	1×1	$128 \times 35 \times 23$
Conv2	$128 \times 35 \times 23$	128	3×3	1×1	$128 \times 35 \times 23$
Conv3	$128 \times 35 \times 23$	128	3×3	1×1	$128 \times 35 \times 23$
Uconv4	$128 \times 35 \times 23$	64	4×4	2×2	$64 \times 70 \times 46$
Uconv5	$64 \times 70 \times 46$	1	4×4	2×2	$1 \times 140 \times 92$
Slice	$1 \times 140 \times 92$	-	-	-	$1 \times 115 \times 80$

Table A.4: The architecture of the Conv4 feature inverter.

Layer	Input shape	$N_{filters}$	Filter	Stride	Output shape
Conv1	$64 \times 33 \times 21$	64	3×3	1×1	$64 \times 33 \times 21$
Conv2	$64 \times 33 \times 21$	64	3×3	1×1	$64 \times 33 \times 21$
Conv3	$64 \times 33 \times 21$	64	3×3	1×1	$64 \times 33 \times 21$
Uconv4	$64 \times 33 \times 21$	32	4×4	2×2	$32 \times 66 \times 42$
Uconv5	$32 \times 66 \times 42$	1	4×4	2×2	$1 \times 132 \times 84$
Slice	$1 \times 132 \times 84$	-	-	-	$1 \times 115 \times 80$

Table A.5: The architecture of the Conv5 feature inverter.

Layer	Input shape	$N_{filters}$	Filter	Stride	Output shape
Conv1	$64 \times 11 \times 7$	64	3×3	1×1	$64 \times 11 \times 7$
Uconv2	$64 \times 11 \times 7$	32	4×4	2×2	$32 \times 22 \times 14$
Uconv3	$32 \times 22 \times 14$	16	4×4	2×2	$16 \times 44 \times 28$
Uconv4	$16 \times 44 \times 28$	8	4×4	2×2	$8 \times 88 \times 56$
Uconv5	$8 \times 88 \times 56$	1	4×4	2×2	$1 \times 176 \times 112$
Slice	$1 \times 176 \times 112$	-	-	-	$1 \times 115 \times 80$

Table A.6: The architecture of the MP6 feature inverter.

Appendix B

Mel-frequency cepstral coefficients

This appendix describes the steps for extracting the mel-frequency cepstral coefficients (MFCCs) and discusses why inverting the MFCCs (and mel-spectrograms) back to the temporal representation is a lossy procedure.

B.1 MFCC extraction

The extraction of k MFCCs from input audio consists of six steps.

- Given an input audio $x[n]$, where n represents the sample index, extract overlapping audio frames of length l[1]. Apply a window (e.g., Hamming window) to each audio frame to reduce spectral leakage. Finally, group all the windowed audio frames in a $t \times l$ matrix $\mathbf{X}$, where t represents the number of audio frames.

- Compute a $t \times f$ matrix $\mathbf{S}$ (spectrogram), where f represents the number of frequency bins, by computing the discrete Fourier transform (DFT) [Müller, 2015] of each frame in $\mathbf{X}$.

- Compute the $t \times f$ power spectrogram (periodogram)[2] matrix $\mathbf{P}$ by computing the power in each frame (row) of the matrix $\mathbf{S}$.

- Compute the $t \times j$ log-scaled mel-spectrogram matrix $\mathbf{M}$ by first multiplying the power spectrogram $\mathbf{P}$ with the $f \times j$ mel-filterbank matrix $\mathbf{W}$,

[1] Generally, audio frames are of short duration (e.g., for the speech signal, audio frame duration is around 20-40 ms).

[2] Some works compute the magnitude spectrogram instead of the power spectrogram [Logan, 2000].

where the columns in $\mathbf{W}$ correspond to the number of filters j, and then computing the logarithm of the resulting product. The mel-filterbank consists of triangular filters with centre frequencies following the perceptual mel-scale, that is linear below 1000 Hz and logarithmic for higher frequencies. Thus, a mel-spectrogram provides more resolution for lower frequencies than higher frequencies. The mel-scaling step aims to highlight the perceptually important frequencies as the perceived pitch follows a non-linear scale. Similarly, the log-scaling relates to the observation that the perceived loudness of an audio signal is approximately logarithmic [Logan, 2000].

- Compute the $t \times j$ MFCC matrix $\mathbf{F}$ by multiplying the log-scaled mel-spectrogram matrix $\mathbf{M}$ by the $j \times j$ discrete cosine transform (DCT) matrix[3] $\mathbf{\Psi}$ [Oppenheim et al., 1999]. This step aims to decorrelate the mel-spectral vectors (a row in $\mathbf{M}$) by multiplying each one of them with the j DCT basis functions.

- Finally, generate a $t \times k$ matrix $\tilde{\mathbf{F}}$ by selecting k lower-order coefficients from each audio frame in $\mathbf{F}$. Each row in $\tilde{\mathbf{F}}$ contains k MFCCs per audio frame.

Formally, the computation of the MFCC matrix $\mathbf{F}$ is given by

$$\mathbf{P} = |\Phi(\mathbf{X})|^2 \tag{B.1}$$

$$\mathbf{M} = \log_a(\mathbf{PW}) \tag{B.2}$$

$$\mathbf{F} = \mathbf{M\Psi} \tag{B.3}$$

$$\tilde{\mathbf{F}} = (f_{bc})_{\substack{1 \leq b \leq t \\ 1 \leq c \leq k}} \tag{B.4}$$

where Φ represents the frame-wise DFT function, $|.|$ represents the absolute value function, and f_{bc} represents an element of $\mathbf{F}$ in the b_{th} row and the c_{th} column.

B.2 MFCC inversion

The inversion of MFCCs back to the temporal representation involves the steps mentioned below.

- The first step aims to map the $t \times k$ matrix $\tilde{\mathbf{F}}$ with each row representing k MFCCs in an audio frame, back to the log-scaled mel-spectrogram matrix $\mathbf{M}$. This involves two sub-steps: the first aims to map $\tilde{\mathbf{F}}$ to $\mathbf{F}$, and the

[3]DCT Type-II and DCT Type-III are the popular choices.

second aims to map $\mathbf{F}$ to $\mathbf{M}$. We can rewrite the $\tilde{\mathbf{F}}$ computation step by combining Eq. B.3 and Eq. B.4 as

$$\tilde{\mathbf{F}} = \mathbf{M}\tilde{\mathbf{\Psi}} \tag{B.5}$$

where $\tilde{\mathbf{\Psi}}$ represents the $j \times k$ DCT matrix with k cosine basis functions. $\mathbf{\Psi}$ is invertible, however, as $\tilde{\mathbf{\Psi}}$ is not a square matrix, its inverse $\tilde{\mathbf{\Psi}}^{-1}$ does not exist. However, we can approximate $\tilde{\mathbf{\Psi}}^{-1}$ by the $k \times j$ Moore-Penrose pseudoinverse $\tilde{\mathbf{\Psi}}_{\mathbf{p}}$ given by [Boucheron and De Leon, 2008]

$$\tilde{\mathbf{\Psi}}_{\mathbf{p}} = (\tilde{\mathbf{\Psi}}^{\top}\tilde{\mathbf{\Psi}})^{-1}\tilde{\mathbf{\Psi}}^{\top} \tag{B.6}$$

We can use $\tilde{\mathbf{\Psi}}_{\mathbf{p}}$ to invert $\tilde{\mathbf{F}}$ to the $t \times j$ mel-spectrogram matrix $\tilde{\mathbf{M}}$ as

$$\tilde{\mathbf{M}} = \tilde{\mathbf{F}}\tilde{\mathbf{\Psi}}_{\mathbf{p}} \tag{B.7}$$

$$\tilde{\mathbf{M}} = \mathbf{M}\tilde{\mathbf{\Psi}}\tilde{\mathbf{\Psi}}_{\mathbf{p}} \tag{B.8}$$

$$\tilde{\mathbf{M}} \approx \mathbf{M} \tag{B.9}$$

Thus, the inverted mel-spectrogram $\tilde{\mathbf{M}}$ differs from the original mel-spectrogram $\mathbf{M}$. However, as for $j = k$, $\tilde{\mathbf{M}} = \mathbf{M}$, we can achieve a lossless inversion by preserving the MFCCs per audio frame for $k + 1 \leq c \leq j$, where c represents the column index in $\mathbf{F}$.

- The second step aims to invert $\tilde{\mathbf{M}}$ to the power spectrogram matrix $\mathbf{P}$.

$$\tilde{\mathbf{M}} \approx \log_a(\mathbf{PW}) \tag{B.10}$$

$$\mathbf{M}' = a^{\tilde{\mathbf{M}}} \approx \mathbf{PW} \tag{B.11}$$

The logarithm is an invertible function, thus, no information loss happens in mapping $\tilde{\mathbf{M}}$ to a $t \times j$ matrix $\mathbf{M}'$. However, similar to the DCT matrix in step 1, the filterbank matrix $\mathbf{W}$ is not invertible. Thus, using its $j \times f$ Moore-Penrose pesudoinverse matrix $\mathbf{W}_{\mathbf{p}}$, we can invert $\mathbf{M}'$ as

$$\mathbf{M}' \approx \mathbf{PW} \tag{B.12}$$

$$\mathbf{M}'\mathbf{W}_{\mathbf{p}} \approx \mathbf{PWW}_{\mathbf{p}} \tag{B.13}$$

$$\tilde{\mathbf{P}} = \mathbf{M}'\mathbf{W}_{\mathbf{p}} \approx \mathbf{P} \tag{B.14}$$

where $\mathbf{W}_{\mathbf{p}} = (\mathbf{W}^{\top}\mathbf{W})^{-1}\mathbf{W}^{\top}$. Thus, the mel-spectrogram inversion step generates $\tilde{\mathbf{P}}$ that is different from the original periodogram matrix $\mathbf{P}$. Moreover, as discussed above, we can prevent the information loss in the DCT inversion step (by keeping $k = j$), but the mel-spectrogram inver-

sion step is lossy due to the grouping of frequency bins according to the perceptual mel-scale.

- The third step inverts the power spectrogram $\tilde{\mathbf{P}}$ to the magnitude spectrogram $|\tilde{\mathbf{S}}|$, where $\tilde{\mathbf{S}}$ represents the $t \times f$ reconstructed spectrogram matrix. This step does not result in any information loss as square is an invertible function, however, $\tilde{\mathbf{P}} \approx \mathbf{P}$ results in $|\tilde{\mathbf{S}}| \approx |\mathbf{S}|$.

- Finally, the last step maps $|\tilde{\mathbf{S}}|$ to the time domain representation $\tilde{x}[n]$. To do that, the method requires the phase spectrogram matrix $\underline{\angle\tilde{\mathbf{S}}}$. The method combines $\underline{\angle\tilde{\mathbf{S}}}$ with $|\tilde{\mathbf{S}}|$ to generate the reconstructed spectrogram matrix $\tilde{\mathbf{S}}$ ($\tilde{\mathbf{S}} = |\tilde{\mathbf{S}}|e^{j\underline{\angle\tilde{\mathbf{S}}}}$) that on applying the inverse DFT (IDFT) [Müller, 2015] function generates $\tilde{x}[n]$. IDFT generates a $t \times l$ matrix $\tilde{\mathbf{X}}$ with each row containing a temporal audio frame of length l samples. The overlap-add method later combines the reconstructed audio frames in $\tilde{\mathbf{X}}$ to generate the temporal audio signal. IDFT is an invertible function, however, whether any information loss happens in the inversion of the reconstructed spectrogram to the temporal representation depends on the phase spectrogram. In the context of this thesis, there are two ways to get $\underline{\angle\tilde{\mathbf{S}}}$: by preserving the phase spectrogram of $x[n]$ and reusing it during the inversion step, i.e., $\underline{\angle\tilde{\mathbf{S}}} = \underline{\angle\mathbf{S}}$ (see Chapter 4), or by synthesising the phase spectrogram using the inverted magnitude spectrogram $|\tilde{\mathbf{S}}|$ (see Chapter 5). To synthesise the phase spectrogram, this thesis uses the Griffin-Lim algorithm [Griffin and Lim, 1984] that starting from a random phase spectrogram searches for the optimal phase spectrogram by minimising the mean square error between the modified magnitude spectrogram $|\tilde{\mathbf{S}}|$ and the estimated magnitude spectrogram $|\tilde{\mathbf{S}}_\mathbf{i}|$ in the i_{th} iteration.

Thus, inverting an MFCC or a mel-spectrogram back to the temporal audio representation involves information loss. We can reduce the loss by preserving the original phase spectrogram and by keeping the number of MFCCs per audio frame and the number of DCT basis functions equal. However, the mel-spectrogram computation step results in unpreventable information loss due to the grouping of frequency bins according to the mel-scale.

Fig. B.1 assists in analysing the information loss associated with the mel-spectrogram inversion and phase spectrogram reconstruction steps. Fig. B.1 (a) depicts power spectrogram of a 2 seconds excerpt from the Jamendo test dataset. The power spectrogram computation uses $l = 1024$ samples and hop length = 315 samples. Fig. B.1 (b) depicts power spectrogram of the signal reconstructed using magnitude spectrogram of the input excerpt. To plot this, the experiment first uses the Griffin-Lim and IDFT algorithms to map magnitude spectrogram

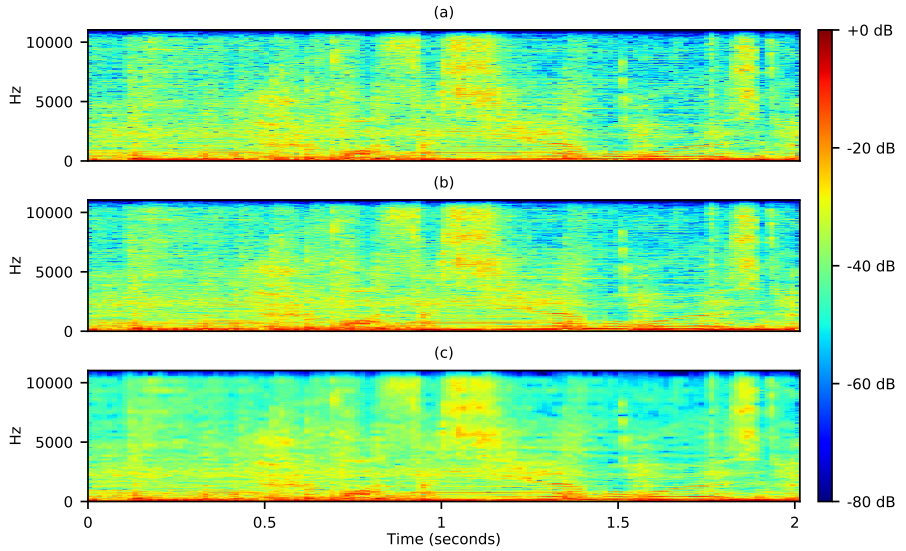

Figure B.1: Visualisations depicting power spectrograms for a 2 seconds audio excerpt from the "05-Elles disent.mp3" file in the Jamendo test dataset (time index: 55.00 seconds - 57.00 seconds). (a) Power spectrogram of the input excerpt, (b) Power spectrogram of the signal reconstructed using magnitude spectrogram of the input excerpt, and (c) Power spectrogram of signal reconstructed using mel-spectrogram of the input excerpt.

of the input excerpt to a temporal signal[4] and then computes power spectrogram of the temporal signal. Finally, Fig. B.1(c) depicts power spectrogram of the temporal signal reconstructed using mel-spectrogram of the input excerpt. To plot this, the experiment first generates mel-spectrogram from the input excerpt using a filterbank of 80 mel-filters logarithmically distributed between 0 Hz to 11025 Hz and then follows the mel-spectrogram inversion steps to generate a temporal signal whose power spectrogram is depicted in Fig. B.1(c).

The visualisation of the power spectrograms and auralisation of the temporal signals suggests that the phase reconstruction step although lossy results in very minimal information loss (Fig. B.1(a) and (b) are nearly indistinguishable). However, the mel-spectrogram inversion process involves information loss, especially for higher frequencies and results in audible artefacts in the inverted temporal signal. This happens due to the design of mel-filterbank that provides more frequency resolution at lower frequencies than at higher frequencies. Interestingly, despite the information loss in inverting mel-spectrograms, auralisation of the inverted temporal signal reveals that the presence of artefacts has limited effect on the interpretability of the signal.

[4]The experiment discards the phase spectrogram of the input excerpt.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv e-prints*, arXiv:1603.04467, 2016.

Julius Adebayo, Justin Gilmer, Michael Muelly, Ian J. Goodfellow, Moritz Hardt, and Been Kim. Sanity Checks for Saliency Maps. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, pages 9525–9536. Montréal, Canada, December 3–8 2018.

Pulkit Agrawal, Ross B. Girshick, and Jitendra Malik. Analyzing the Performance of Multilayer Neural Networks for Object Recognition. In *Proceedings of the 13th European Conference on Computer Vision (ECCV)*, pages 329–344. Zurich, Switzerland, September 6–12 2014.

Michal Aharon, Michael Elad, and Alfred Bruckstein. K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006.

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre-Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron Courville,

Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Mélanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian Goodfellow, Matt Graham, Caglar Gulcehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrancois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert T. McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A Python Framework for Fast Computation of Mathematical Expressions. *arXiv e-prints*, arXiv:1605.02688, May 2016.

David Alvarez-Melis and Tommi S. Jaakkola. Towards Robust Interpretability with Self-Explaining Neural Networks. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, pages 7786–7795. Montréal, Canada, December 3–8 2018.

Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. Towards Better Understanding of Gradient-Based Attribution Methods for Deep Neural Networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*. Vancouver, Canada, April 30–May 3 2018.

Marco Ancona, Cengiz Öztireli, and Markus Gross. Explaining Deep Neural Networks with a Polynomial Time Algorithm for Shapley Value Approximation. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 272–281. Long Beach, California, USA, June 9–15 2019.

Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine Bias, May 2016. URL `https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing`. Accessed October 8, 2019.

Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 214–223. Sydney, Australia, August 6–11 2017.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, and Klaus-Robert Müller. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PloS one*, 10(7), 2015.

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How To Explain Individual Classification Decisions. *Journal of Machine Learning Research*, 11:1803–1831, 2010.

Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in Optimizing Recurrent Networks. In *Proceedings of the 38th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8624–8628. Vancouver, Canada, May 26–31 2013.

Adam L. Berenzweig and Daniel P. W. Ellis. Locating Singing Voice Segments within Music Signals. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 119–122. New Paltz, New York, USA, October 21–24 2001.

Adam L. Berenzweig, Daniel P. W. Ellis, and Steve Lawrence. Using Voice Segments to Improve Artist Classification of Music. In *Proceedings of the 22nd Audio Engineering Society (AES) Conference: Virtual, Synthetic, and Entertainment Audio*. Espoo, Finland, June 2002.

Christopher M. Bishop. *Pattern Recognition and Machine Learning, 5th Edition*. Information Science and Statistics. Springer, 2007.

Rachel M. Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, pages 155–160. Taipei, Taiwan, October 2014.

Laura Boucheron and Phillip De Leon. On the Inversion of Mel-Frequency Cepstral Coefficients for Speech Enhancement Applications. In *Proceedings of the IEEE International Conference on Signals and Electronic Systems (ICSES)*, pages 485–488. Krakow, Poland, September 14–17 2008.

Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

Vicente Iván Sánchez Carmona, Tim Rocktäschel, Sebastian Riedel, and Sameer Singh. Towards Extracting Faithful and Descriptive Representations of Latent Variable Models. In *AAAI Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches*. Palo Alto, California, USA, March 23–25 2015.

Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. Activation Atlas. *Distill*, 2019.

Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible Models for Healthcare: Predicting Pneumonia Risk and Hospital 30-day Readmission. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1721–1730. Sydney, Australia, August 10–13 2015.

Chaofan Chen, Oscar Li, Alina Barnett, Jonathan Su, and Cynthia Rudin. This Looks Like That: Deep Learning for Interpretable Image Recognition. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*. Vancouver, Canada, December 8–14 2019.

Bhusan Chettri, Saumitra Mishra, Bob L. Sturm, and Emmanouil Benetos. Analysing the Predictions of a CNN-Based Replay Spoofing Detection System. In *Proceedings of the IEEE Spoken Language Technology Workshop (SLT)*, pages 92–97. Athens, Greece, December 18–21 2018.

Keunwoo Choi, György Fazekas, and Mark B. Sandler. Explaining Deep Convolutional Neural Networks on Music Classification. *arXiv e-prints*, arXiv:1607.02444, 2016.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*. San Juan, Puerto Rico, May 2016.

Mark W. Craven and Jude W. Shavlik. Extracting Tree-Structured Representations of Trained Networks. In *Proceedings of the 8th Conference on Neural Information Processing Systems (NeurIPS)*, pages 24–30. Denver, Colorado, USA, November 27–30 1995.

Steven B Davis and Paul Mermelstein. Comparison of Parametric Representation for Monosyllabic Word Recognition in Continously Spoken Sentences.

IEEE Transactions on Acoustic, Speech and Signal Processing, 28(4):357–366, 1980.

Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. FMA: A Dataset for Music Analysis. In *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR)*, pages 316–323. Suzhou, China, October 23–27 2017.

Alexandre Défossez, Neil Zeghidour, Nicolas Usunier, Léon Bottou, and Francis Bach. SING: Symbol-to-Instrument Neural Generator. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, pages 9055–9065. Montréal, Canada, December 3–8 2018.

Sander Dieleman and Benjamin Schrauwen. End-to-End Learning for Music Audio. In *Proceedings of the 39th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6964–6968. Florence, Italy, May 4–9 2014.

Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jeffrey De Fauw, Michael Heilman, Diogo Moitinho de Almeida, Brian McFee, Hendrik Weideman, Gábor Takács, Peter de Rivaz, Jon Crall, Gregory Sanders, Kashif Rasul, Cong Liu, Geoffrey French, and Jonas Degrave. Lasagne: First release, August 2015. URL `http://dx.doi.org/10.5281/zenodo.27878`.

Finale Doshi-Velez and Been Kim. Towards a Rigorous Science of Interpretable Machine Learning. *arXiv e-prints*, arXiv:1702.08608, 2017.

Alexey Dosovitskiy and Thomas Brox. Inverting Visual Representations with Convolutional Networks. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4829–4837. Las Vegas, USA, June 2016a.

Alexey Dosovitskiy and Thomas Brox. Generating Images with Perceptual Similarity Metrics Based on Deep Networks. In *Proceedings of the 30th Conference on Neural Information Processing Systems (NeurIPS)*, pages 658–666. Barcelona, Spain, December 2016b.

Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to Generate Chairs with Convolutional Neural Networks. In *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1538–1546. Boston, USA, June 2015.

Alexey Dosovitskiy, Jost Tobias Springenberg, Maxim Tatarchenko, and Thomas Brox. Learning to Generate Chairs, Tables and Cars with Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):692–705, 2017.

Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for Interpretable Machine Learning. *arXiv e-prints*, arXiv:1808.00033, 2018a.

Mengnan Du, Ninghao Liu, Qingquan Song, and Xia Hu. Towards Explanation of DNN-based Prediction with Guided Feature Inversion. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1358–1367. London, UK, August 2018b.

Jesse Engel, Lamtharn (Hanoi) Hantrakul, Chenjie Gu, and Adam Roberts. DDSP: Differentiable Digital Signal Processing. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*. Addis Ababa, Ethiopia, April 26–May 1 2020.

Dumitru Erhan, Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Visualising Higher-Layer Features of a Deep Network. Technical Report 1341, University of Montreal, June 2009.

J. L. Fleiss. Measuring Nominal Scale Agreement Among Many Raters. *Psychological Bulletin*, 76(5):378–382, 1971.

Arthur Flexer and Dominik Schnitzer. Effects of Album and Artist Filters in Audio Similarity Computed for Very Large Music Databases. *Computer Music Journal*, 34(3):20–28, 2010.

Ruth C. Fong and Andrea Vedaldi. Interpretable Explanations of Black Boxes by Meaningful Perturbation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 3449–3457. Venice, Italy, October 22–29 2017.

James R. Foulds and Eibe Frank. A Review of Multi-Instance Learning Assumptions. *Knowledge Engineering Review*, 25(1):1–25, 2010.

Alex Alves Freitas. Comprehensible Classification Models: A Position Paper. *SIGKDD Explorations*, 15(1):1–10, 2013.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 1050–1059. New York, USA, June 19–24 2016.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423. Las Vegas, USA, June 2016.

Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining Explanations: An Overview of Interpretability of Machine Learning. In *Proceedings of the 5th IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 80–89. Turin, Italy, October 1–3 2018.

Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proceedings of the 27th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587. Columbus, USA, June 23–28 2014.

Ian J. Goodfellow, Quoc V. Le, Andrew M. Saxe, Honglak Lee, and Andrew Y. Ng. Measuring Invariances in Deep Networks. In *Proceedings of the 23rd Conference on Neural Information Processing Systems (NeurIPS)*, pages 646–654. Vancouver, British Columbia, Canada, December 7-10 2009.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative Adverserial Nets. In *Proceedings of the 28th Conference on Neural Information Processing Systems (NeurIPS)*, pages 2672–2680. Montréal, Quebec, Canada, December 8–13 2014.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adverserial Examples. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. San Diego, USA, May 7–9 2015.

Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive Computation and Machine Learning. MIT Press, 2016.

Bryce Goodman and Seth R. Flaxman. European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation". *AI Magazine*, 38(3): 50–57, 2017.

Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. RWC Music Database: Popular, Classical and Jazz Music Databases. In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR)*, pages 287–288. Paris, France, October 2002.

Daniel W. Griffin and Jae S. Lim. Signal Estimation From Modified Short-Time Fourier Transform. *IEEE Transactions on Acoustic, Speech, and Signal Processing*, 32(2):236–243, 1984.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A Survey of Methods for Explaining Black Box Models. *ACM Computing Surveys*, 51(5):93:1–93:42, 2019.

Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved Training of Wasserstein GANs. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS)*, pages 5767–5777. Long Beach, California, USA, December 4–9 2017.

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer, 2nd edition, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034. Santiago, Chile, December 2015.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS)*, pages 6626–6637. Long Beach, California, USA, December 4–9 2017.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. *arXiv e-prints*, arXiv:1207.0580, July 2012.

R. Devon Hjelm, Athul Paul Jacob, Tong Che, Kyunghyun Cho, and Yoshua Bengio. Boundary-Seeking Generative Adversarial Networks. *arXiv e-prints*, arXiv:1702.08431, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Eric J. Humphrey, Juan Pablo Bello, and Yann LeCun. Moving Beyond Feature Design: Deep Architectures and Automatic Feature Learning in Music Informatics. In *Proceedings of the 13th International Society for Music Information Retrieval Conference (ISMIR)*, pages 403–408. Porto, Portugal, October 8-12 2012.

Eric J. Humphrey, Sravana Reddy, Prem Seetharaman, Aparna Kumar, Rachel M. Bittner, Andrew Demetriou, Sankalp Gulati, Andreas Jansson, Tristan Jehan, Bernhard Lehner, Anna Kruspe, and Luwei Yang. An Introduction to Signal Processing for Singing-Voice Analysis: High Notes in the Effort to Automate the Understanding of Vocals in Music. *IEEE Signal Processing Magazine*, 36(1):82–94, 2019.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 448–456. Lille, France, July 2015.

Jesper Højvang Jensen, Mads Græsbøll Christensen, Daniel P. W. Ellis, and Søren Holdt Jensen. Quantitative Analysis of a Common Audio Similarity Measure. *IEEE Transactions on Audio, Speech, and Langauage Processing*, 17(4):693–703, 2009.

Dan Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall Series in Artificial Intelligence. Prentice Hall, Pearson Education International, 2nd edition, 2009.

Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and Understanding Recurrent Networks. *arXiv e-prints*, arXiv:1506.02078, 2015.

Corey Kereliuk, Bob L. Sturm, and Jan Larsen. Deep Learning, Audio Adversaries, and Music Content Analysis. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 1–5. New Paltz, New York, USA, October 18–21 2015.

Been Kim, Cynthia Rudin, and Julie A. Shah. The Bayesian Case Model: A Generative Approach for Case-Based Reasoning and Prototype Classification. In *Proceedings of the 28th Conference on Neural Information Processing Systems (NeurIPS)*, Montreal, Quebec, Canada, pages 1952–1960, December 8–13 2014.

Been Kim, Oluwasanmi Koyejo, and Rajiv Khanna. Examples Are Not Enough, Learn to Criticize! Criticism for Interpretability. In *Proceedings of the 30th Conference on Neural Information Processing Systems (NeurIPS)*, pages 2280–2288. Barcelona, Spain, December 5–10 2016.

Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T. Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (Un)reliability of Saliency Methods. *arXiv e-prints*, arXiv:1711.00867, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. San Diego, USA, May 2015.

Pang Wei Koh and Percy Liang. Understanding Black-Box Predictions via Influence Functions. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1885–1894. Sydney, Australia, August 6–11 2017.

Josua Krause, Adam Perer, and Kenney Ng. Interacting With Predictions: Visual Inspection of Black-Box Machine Learning Models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 5686–5697. San Jose, USA, May 7–12 2016.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 26th Conference on Neural Information Processing Systems (NeurIPS)*, pages 1106–1114. Lake Tahoe, USA, December 2012.

Andreas Krug and Sebastian Stober. Introspection for Convolutional Automatic Speech Recognition. In *Proceedings of the EMNLP Workshop on Analysing and Interpreting Neural Networks for NLP*, pages 187–199. Brussels, Belgium, November 1 2018.

Andreas Krug, René Knaebel, and Sebastian Stober. Neuron Activation Profiles for Interpreting Convolutional Speech Recognition Models. In *Proceedings of the NeurIPS Workshop on Interpretability and Robustness in Audio, Speech and Language*. Montréal, Canada, December 8 2018.

Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable Decision Sets: A Joint Framework for Description and Prediction. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1675–1684. San Francisco, California, USA, August, 13–17 2016.

J. R. Landis and G. G. Koch. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33(1):159–174, 1977.

Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking Clever Hans Predictors and Assessing What Machines Really Learn. *Nature Communications*, 10(1):1096, 2019.

Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Greg Corrado, Kai Chen, Jeffrey Dean, and Andrew Y. Ng. Building High-Level Features Using Large Scale Unsupervised Learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*. Edinburg, Scotland, UK, June 26–July 1 2012.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep Learning. *Nature*, 521(7553):436 – 444, 2015.

Kyungyun Lee, Keunwoo Choi, and Juhan Nam. Revisiting Singing Voice Detection: A Quantitative Review and the Future Outlook. In *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, pages 506–513. Paris, France, September 23–27 2018.

Simon Leglaive, Romain Hennequin, and Roland Badeau. Singing Voice Detection with Deep Recurrent Neural Networks. In *Proceedings of the 40th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 121–125. South Brisbane, Queensland, Australia, April 2015.

Bernhard Lehner, Reinhard Sonnleitner, and Gerhard Widmer. Towards Light-Weight, Real-Time-Capable Singing Voice Detection. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR)*, pages 53–58. Curitiba, Brazil, November 2013.

Bernhard Lehner, Gerhard Widmer, and Reinhard Sonnleitner. On the Reduction of False Positives in Singing Voice Detection. In *Proceedings of the 39th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7480–7484. Florence, Italy, May 2014.

Bernhard Lehner, Gerhard Widmer, and Sebastian Böck. A Low-Latency, Real-Time-Capable, Singing Voice Detection Method with LSTM Recurrent Neural Networks. In *Proceedings of the 23rd European Signal Processing Conference (EUSIPCO)*, pages 21–25. Nice, France, August 2015.

Bernhard Lehner, Jan Schlüter, and Gerhard Widmer. Online, Loudness-Invariant Vocal Detection in Mixed Music Signals. *IEEE/ACM Transactions on Audio, Speech, and Langauage Processing*, 26(8):1369–1380, 2018.

Tao Lei, Regina Barzilay, and Tommi S. Jaakkola. Rationalizing Neural Predictions. In *Proceedings of the Conference on Empirical Methods in Natural*

Language Processing (EMNLP), pages 107–117. Austin, Texas, November 1–5 2016.

Benjamin Letham, Cynthia Rudin, Tyler H. McCormick, and David Madigan. Interpretable Classifiers Using Rules and Bayesian Analysis: Building a Better Stroke Prediction Model. *The Annals of Applied Statistics*, 9(3):1350–1371, 2015.

Jiwei Li, Xinlei Chen, Eduard H. Hovy, and Dan Jurafsky. Visualizing and Understanding Neural Models in NLP. In *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT)*, pages 681–691. San Diego, California, USA, June 12–17 2016.

Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. Deep Learning for Case-Based Reasoning Through Prototypes: A Neural Network That Explains Its Predictions. In *Proceedings of the 32nd Conference on Artificial Intelligence (AAAI)*, pages 3530–3537. New Orleans, Louisiana, USA, February 2–7 2018.

R. Likert. A Technique For The Measurement Of Attitudes. *Archives of Psychology*, 1932.

Zachary C. Lipton. The Mythos of Model Interpretability. In *Proceedings of the International Conference on Machine Learning (ICML) Workshop on Human Interpretability in Machine Learning*. New York, USA, June 2016.

Antoine Liutkus, Derry Fitzgerald, Zafar Rafii, Bryan Pardo, and Laurent Daudet. Kernel Additive Models for Source Separation. *IEEE Transactions on Signal Processing*, 62(16):4298–4310, 2014.

Beth Logan. Mel Frequency Cepstral Coefficients for Music Modeling. In *Proceedings of the 1st International Symposium for Music Information Retrieval (ISMIR)*. Plymouth, Massachusetts, USA, October 23–25 2000.

Scott M. Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS)*, pages 4765–4774. Long Beach, California, USA, December 4–9 2017.

Aravindh Mahendran and Andrea Vedaldi. Understanding Deep Image Representations by Inverting Them. In *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5188–5196. Boston, USA, June 2015.

Aravindh Mahendran and Andrea Vedaldi. Visualizing Deep Convolutional Neural Networks Using Natural Pre-images. *International Journal of Computer Vision*, 120(3):233–255, 2016.

Andrew L. Mass, Awni Y. Hannun, and Andrew Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *Proceedings of the ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. Atlanta, USA, June 2013.

Matthias Mauch, Hiromasa Fujihara, Kazuyoshi Yoshii, and Masataka Goto. Timbre and Melody Features for the Recognition of Vocal Activity and Instrumental Solos in Polyphonic Music. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, pages 233–238. Miami, Florida, USA, October 2011.

Brian McFee, Colin Raffel, Dawen Liang, Daniel P. W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. Librosa: Audio and Music Signal Analysis in Python. In *Proceedings of the 14th Python in Science Conference (SCIPY)*, pages 18–25. Austin, Texas, USA, July 2015.

Luke Merrick and Ankur Taly. The Explanation Game: Explaining Machine Learning Models with Cooperative Game Theory. *arXiv e-prints*, arXiv:1909.08128, 2019.

Tim Miller. Explanation in Artificial Intelligence: Insights from the Social Sciences. *Artificial Intelligence*, 267:1–38, 2019.

Saumitra Mishra, Bob L. Sturm, and Simon Dixon. Local Interpretable Model-Agnostic Explanations for Music Content Analysis. In *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR)*, pages 537–543. Suzhou, China, October 2017.

Saumitra Mishra, Bob L. Sturm, and Simon Dixon. "What are You Listening to?" Explaining Predictions of Deep Machine Listening Systems. In *Proceedings of the 26th European Signal Processing Conference*, pages 2260–2264, Rome, Italy, September 2018a.

Saumitra Mishra, Bob L. Sturm, and Simon Dixon. Understanding a Deep Machine Listening Model Through Feature Inversion. In *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, pages 755–762, Paris, France, September 2018b.

Saumitra Mishra, Daniel Stoller, Emmanouil Benetos, Bob L. Sturm, and Simon Dixon. GAN-based Generation and Automatic Selection of Explanations for

Neural Networks. In *Proceedings of the International Conference on Learning Representations (ICLR) Workshop on Safe Machine Learning*. New Orleans, USA, May 6–9 2019.

Saumitra Mishra, Emmanouil Benetos, Bob L. Sturm, and Simon Dixon. Reliable Local Explanations for Machine Listening. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN) Special Session on Explainable Computational/Artificial Intelligence*. Glasgow, Scotland, July 19–24 2020.

Brent D. Mittelstadt, Chris Russell, and Sandra Wachter. Explaining Explanations in AI. In *Proceedings of the 2nd ACM Conference on Fairness, Accountability, and Transparency (FAT*)*, pages 279–288. Atlanta, Georgia, USA, January 29–31 2019.

Christoph Molnar. Interpretable Machine Learning: A Guide for Making Black Box Models Explainable, 2019. URL `https://christophm.github.io/interpretable-ml-book/`. Accessed December 17, 2019.

Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for Interpreting and Understanding Deep Neural Networks. *Digital Signal Processing*, 73:1–15, 2018.

Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going Deeper into Neural Networks, June 2015. URL `https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`. Accessed October 15, 2019.

Meinard Müller. *Fundamentals of Music Processing - Audio, Analysis, Algorithms, Applications*. Springer, 2015.

W. James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Interpretable Machine Learning: Definitions, Methods, and Applications. *arXiv e-prints*, arXiv:1901.04592, 2019.

Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436. Boston, USA, June 2015.

Anh Mai Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the Preferred Inputs for Neurons in Neural Networks Via Deep Generator Networks. In *Proceedings of the 30th Conference on Neural Information Processing Systems (NeurIPS)*, pages 3387–3395. Barcelona, Spain, December 5–10 2016a.

Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks. In *Proceedings of the International Conference on Machine Learning (ICML) Workshop on Visualization for Deep Learning*. New York, USA, June 2016b.

Anh Mai Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space. In *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3510–3520. Honolulu, USA, July 21–26 2017.

Adam Noack, Isaac Ahern, Dejing Dou, and Boyang Li. Does Interpretability of Neural Networks Imply Adversarial Robustness? *arXiv e-prints*, arXiv:1912.03430, 2019.

Tin Lay Nwe and Haizhou Li. Singing Voice Detection Using Perceptually-Motivated Features. In *Proceedings of the 15th ACM International Conference on Multimedia (ACMMM)*, pages 309–312. Augsburg, Germany, September 24–29 2007.

Tin Lay Nwe and Ye Wang. Automatic Detection of Vocal Segments in Popular Songs. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*. Barcelona, Spain, October 10–14 2004.

Tin Lay Nwe, Arun Shenoy, and Ye Wang. Singing Voice Detection in Popular Music. In *Proceedings of the 12th ACM International Conference on Multimedia (ACMMM)*, pages 324–327. New York, USA, October, 10–16 2004.

Christopher Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature Visualization. *Distill*, 2017.

Alan V. Oppenheim, John R. Buck, and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall, 2nd edition, 1999.

Michalis Papakostas and Theodoros Giannakopoulos. Speech-Music Discrimination Using Deep Visual Feature Extractors. *Expert System with Applications*, 114:334–344, 2018.

Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Machine Learning. In *Proceedings of the ACM on Asia Conference on Computer and Communications Security (AsiaCCS)*, pages 506–519. Abu Dhabi, United Arab Emirates, April 2–6 2017.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Karol J. Piczak. Environmental Sound Classification with Convolutional Neural Networks. In *Proceedings of the 25th IEEE International Workshop on Machine Learning for Signal Processing*, pages 1–6. Boston, USA, September 17–20 2015.

Aggelos Pikrakis, Yannis Kopsinis, Nadine Kroher, and José Miguel Díaz-Báñez. Unsupervised Singing Voice Detection Using Dictionary Learning. In *Proceedings of the 24th European Signal Processing Conference (EUSIPCO)*, pages 1212–1216. Budapest, Hungary, August 29–September 2 2016.

Jordi Pons, Oriol Nieto, Matthew Prockup, Erik M. Schmidt, Andreas F. Ehmann, and Xavier Serra. End-to-end Learning for Music Audio Tagging at Scale. In *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, pages 637–644. Paris, France, September 23–27 2018.

R. Quian Quiroga, L. Reddy, G. Kreiman, C. Koch, and I. Fried. Invariant Visual Representation by Single Neurons in the Human Brain. *Nature*, 435 (7045):1102–1107, 2005.

Lawrence Rabiner and Ronald Schafer. *Theory and Applications of Digital Speech Processing*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2010.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*. San Juan, Puerto Rico, May 2–4 2016.

Mathieu Ramona, Gaël Richard, and Bertrand David. Vocal Detection in Music Using Support Vector Machines. In *Proceedings of the 33rd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1885–1888. Las Vegas, Nevada, USA, April 2008.

Lise Regnier and Geoffroy Peeters. Singing Voice Detection in Music Tracks Using Direct Voice Vibrato Detection. In *Proceedings of the 34th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1685–1688. Taipei, Taiwan, April 19–24 2009.

Jan Rennies and Henning Schepker. Listening Effort and Speech Intelligibility in Listening Situations Affected by Noise and Reverberation. *The Journal of the Acoustical Society of America*, 136(5), 2014.

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. Model Agnostic Interpretability of Machine Learning. In *Proceedings of the International Conference on Machine Learning (ICML) Workshop on Human Interpretability in Machine Learning*. New York, USA, June 2016a.

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1135–1144. San Francisco, USA, August 2016b.

Martín Rocamora and Perfecto Herrera. Comparing Audio Descriptors for Singing Voice Detection in Music Audio Files. In *11º Simpósio Brasileiro de Computação Musical (SBCM)*. Sao Paulo, Brazil, September 2007.

Francisco Rodríguez-Algarra, Bob L. Sturm, and Hugo Maruri-Aguilar. Analysing Scattering-Based Music Content Analysis Systems: Where's the Music? In *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*, pages 344–350. New York, USA, August 7–11 2016.

Cynthia Rudin. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-Propagating Errors. *Nature*, 323(6088):533–536, 1986.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. Evaluating the Visualization of What a Deep Neural Network has Learned. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2660–2673, November 2017.

Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller, editors. *Explainable AI: Interpreting, Explaining and*

Visualizing Deep Learning, volume 11700 of *Lecture Notes in Computer Science*. Springer, 2019.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*. Banff, Canada, April 2014.

Jan Schlüter. Learning to Pinpoint Singing Voice from Weakly Labeled Examples. In *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*, pages 44–50. New York, USA, August 7–11 2016.

Jan Schlüter. *Deep Learning for Event Detection, Sequence Labelling and Similarity Estimation in Music Signals*. PhD thesis, Johannes Kepler University Linz, July 2017.

Jan Schlüter and Sebastian Böck. Improved Musical Onset Detection with Convolutional Neural Networks. In *Proceedings of the 39th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6979–6983. Florence, Italy, May 4–9 2014.

Jan Schlüter and Thomas Grill. Exploring Data Augmentation for Improved Singing Voice Detection with Neural Networks. In *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*, pages 121–126. Málaga, Spain, October 2015.

Jan Schlüter and Bernhard Lehner. Zero-Mean Convolutions for Level-Invariant Singing Voice Detection. In *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, pages 321–326. Paris, France, September 23–27 2018.

Jan Schlüter and Reinhard Sonnleitner. Unsupervised Feature Learning for Speech and Music Detection in Radio Broadcasts. In *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx)*, pages 369–376. York, UK, September 17–21 2012.

Florian Scholz, Igor Vatolkin, and Günter Rudolph. Singing Voice Detection Across Different Music Genres. In *Proceedings of AES International Conference on Semantic Audio*. Erlangen, Germany, June 22–24 2017.

Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *Proceedings of the*

IEEE International Conference on Computer Vision (ICCV), pages 618–626. Venice, Italy, October 22–29 2017.

Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not Just a Black Box: Learning Important Features Through Propagating Activation Differences. *arXiv preprint*, arXiv:1605.01713, 2016.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. San Diego, USA, May 2015.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR), Workshop Track*. Banff, Canada, April 2014.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. Smoothgrad: Removing Noise by Adding Noise. *arXiv e-prints*, arXiv:1706.03825, 2017.

Jake Snell, Karl Ridgeway, Renjie Liao, Brett D. Roads, Michael C. Mozer, and Richard S. Zemel. Learning to Generate Images with Perceptual Similarity Metrics. In *Proceedings of the 24th IEEE International Conference on Image Processing (ICIP)*, pages 4277–4281. Beijing, China, September 2017.

Kacper Sokol and Peter A. Flach. Counterfactual Explanations of Machine Learning Predictions: Opportunities and Challenges for AI Safety. In *Proceedings of the Workshop on Artificial Intelligence Safety 2019 co-located with the 33rd AAAI Conference on Artificial Intelligence 2019 (AAAI)*. Honolulu, Hawaii, January 27 2019.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for Simplicity: The All Convolutional Net. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR), Workshop Track*. San Diego, USA, May 7-9 2015.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Dan Stowell, Dimitrios Giannoulis, Emmanouil Benetos, Mathieu Lagrange, and Mark D. Plumbley. Detection and Classification of Acoustic Scenes and Events. *IEEE Transactions on Multimedia*, 17(10):1733–1746, 2015.

Erik Strumbelj and Igor Kononenko. An Efficient Explanation of Individual Classifications Using Game Theory. *Journal of Machine Learning Research*, 11:1–18, 2010.

Bob L. Sturm. A Simple Method to Determine if a Music Information Retrieval System is a "Horse". *IEEE Transactions on Multimedia*, 16(6):1636–1644, 2014.

Bob L. Sturm, Marcela Morvidone, and Laurent Daudet. Musical Instrument Identification Using Multiscale Mel-Frequency Cepstral Coefficients. In *Proceedings of the 18th European Signal Processing Conference (EUSIPCO)*, pages 477–481. Aalborg, Denmark, August 23–27 2010.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3319–3328. Sydney, Australia, August 6–11 2017.

Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the Importance of Initialization and Momentum in Deep Learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1139–1147. Atlanta, USA, June 2013.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing Properties of Neural Networks. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*. Banff, Canada, April 14–16 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. In *Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9. Boston, USA, June 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826. Las Vegas, USA, June 26–July 1 2016.

Hideyuki Tachibana, Takuma Ona, Nobutaka Ono, and Shigeki Sagayama. Melody Line Estimation in Homophonic Music Audio Signals Based on Temporal-Variability of Melodic Source. In *Proceedings of the 35th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 425–428. Dallas, Texas, USA, March 14–19 2010.

Sebastian Thrun. Extracting Rules from Artificial Neural Networks with Distributed Representations. In *Proceedings of the 7th Conference on Neural Information Processing Systems (NeurIPS)*, pages 505–512. Denver, Colorado, USA, 1994.

Wei-Ho Tsai and Hsin-Min Wang. Automatic Singer Recognition of Popular Music Recordings via Estimation and Modeling of Solo Vocal Signals. *IEEE Transactions on Audio, Speech, and Langauage Processing*, 14(1):330–341, 2006.

Paul Upchurch, Jacob R. Gardner, Geoff Pleiss, Robert Pless, Noah Snavely, Kavita Bala, and Kilian Q. Weinberger. Deep Feature Interpolation for Image Content Changes. In *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6090–6099. Honolulu, USA, July 2017.

Berk Ustun and Cynthia Rudin. Supersparse Linear Integer Models for Optimized Medical Scoring Systems. *Machine Learning*, 102(3):349–391, 2015.

Shankar Vembu and Stephan Baumann. Separation of Vocals from Polyphonic Audio Recordings. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 337–344. London, UK, September 2005.

Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. *arXiv e-prints*, arXiv:1711.00399, 2017.

Fulton Wang and Cynthia Rudin. Falling Rule Lists. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS)*. San Diego, California, USA, May 9–12 2015.

Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. Bayesian Rule Sets for Interpretable Classification. In *Proceedings of the IEEE 16th International Conference on Data Mining (ICDM)*, pages 1269–1274. Barcelona, Spain, December, 12-15 2016.

Ye Wang, Min-Yen Kan, Tin Lay Nwe, Arun Shenoy, and Jun Yin. LyricAlly: Automatic Synchronization of Acoustic Musical Signals and Textual Lyrics. In *Proceedings of the 12th ACM International Conference on Multimedia (ACMMM)*, pages 212–219. New York, USA, October 2004.

Donglai Wei, Bolei Zhou, Antonio Torralba, and William T. Freeman. Understanding Intra-Class Knowledge Inside CNN. *arXiv e-prints*, arXiv:1507.02379, 2015.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 2048–2057. Lille, France, July 6–11 2015.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How Transferable Are Features in Deep Neural Networks? In *Proceedings of the 28th Conference on Neural Information Processing Systems (NeurIPS)*, pages 3320–3328. Montréal, Quebec, Canada, December 8–13 2014.

Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding Neural Networks Through Deep Visualization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML), Deep Learning Workshop*. Lille, France, July 2015.

Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In *Proceedings of the 13th European Conference on Computer Vision (ECCV)*, pages 818–833. Zurich, Switzerland, September 2014.

Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable Convolutional Neural Networks. In *Proceedings of the 31st IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8827–8836. Salt Lake City, USA, June 18–22 2018.

Yichi Zhang and Zhiyao Duan. Visualization and Interpretation of Siamese Style Convolutional Neural Networks for Sound Search by Vocal Imitation. In *Proceedings of the 43rd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2406–2410. Calgary, Canada, April 15–20 2018.

Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torrabla. Object Detectors Emerge in Deep Scene CNNs. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. San Diego, USA, May 7–9 2015.

Luisa M. Zintgraf, Taco S. Cohen, Tameem Adel, and Max Welling. Visualizing Deep Neural Network Decisions: Prediction Difference Analysis. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. Toulon, France, April 24–26 2017.