UNDERSTANDING A DEEP MACHINE LISTENING MODEL THROUGH FEATURE INVERSION

Saumitra Mishra, Bob L. Sturm, Simon Dixon
Centre for Digital Music, School of Electronic Engineering and Computer Science
Queen Mary University of London, United Kingdom
{ saumitra.mishra, b.sturm, s.e.dixon}@qmul.ac.uk

ABSTRACT

Methods for interpreting machine learning models can help one understand their global and/or local behaviours, and thereby improve them. In this work, we apply a global analysis method to a machine listening model, which essentially inverts the features generated in a model back into an interpretable form like a sonogram. We demonstrate this method for a state-of-the-art singing voice detection model. We train up-convolutional neural networks to invert the feature generated at each layer of the model. The results suggest that the deepest fully connected layer of the model does not preserve temporal and harmonic structures, but that the inverted features from the deepest convolutional layer do. Moreover, a qualitative analysis of a large number of inputs suggests that the deepest layer in the model learns a decision function as the information it preserves depends on the class label associated with an input.

1. INTRODUCTION

Deep neural networks (DNNs) are state-of-the-art in numerous machine learning applications. This success is due to their high expressive power and strong generalisation capability [10]. DNNs acquire these capabilities automatically through training over large amounts of data and tuning of millions of parameters. Despite their success, DNNs remain "black-boxes" as we know very little about the process by which they form their predictions.

Recent research has highlighted problems associated with DNNs. For example, researchers have demonstrated that attacking these models with carefully generated inputs, called "adversarial examples", changes their predictions [11, 15, 44]. Such behaviour may be dangerous to a system (e.g., autonomous vehicle) if its decision making depends on DNN predictions [32]. Moreover, like shallow machine learning models, a DNN may exploit confounders in a dataset and behave correctly for the wrong reasons. Such behaviour limits the performance of a model in the real world where such confounders are absent. Thus, there is an urgent need to bring interpretability to these black-box models, i.e. to understand the behaviour of a DNN [4].

Researchers have attempted to analytically [33, 48] and empirically explain the behaviour of DNNs. In this work, we focus on understanding these models empirically using post-hoc visualisation methods [27, 31]. We can broadly classify such methods into two categories: (1) methods that explain model predictions (local analysis); and (2) methods that explain a model (global analysis). Local analysis uses variants of *sensitivity analysis* (e.g., gradient-based sensitivity analysis) to generate attribution maps that highlight the input dimensions [39, 40, 43] or input regions (groups of contiguous dimensions) [35, 47] in favour of (or against) a prediction. Such analysis is useful but may result in inconsistent [16] and uninterpretable (noisy) explanations [41]. Although some local explanation methods can generate cleaner visualisations, they depend on the type of non-linearity [42] or network architecture [38, 47] and thus are not generalisable.

In another direction, the global analysis of DNNs aims for an insight that generalises across input instances. For example, irrespective of the class label associated with an input image, the shallow layers of image content recognition models show sensitivity to low-level structures, e.g., edges and gradients [47]. There exist several methods for global analysis. For example, *activation maximisation* aims to synthesise examples in the input space (e.g., pixels) that maximally activate a specific neuron [9, 29, 40, 46] or layer [28] in a model. Similarly, *feature inversion* aims to highlight the input content (features) preserved by any layer in a DNN model by inverting the corresponding feature [6, 23].

In this work, we apply feature inversion to a machine listening model that classifies an input audio frame (or excerpt) into predefined classes. Previous work in the analysis of deep machine listening models has focused both on local and global analysis. The methods to generate local explanations for model predictions use bin-level [1] or region-level [26] attribution maps. On the other hand, Dieleman et al. [3] globally analyse a music autotagging model by visualising filters in the first convolutional layer. Similarly, in [36] the authors globally analyse a scaled-down version of their deep onset detector by visualising the most activated feature maps and their corresponding filter kernels. Our method differs from these global analy-

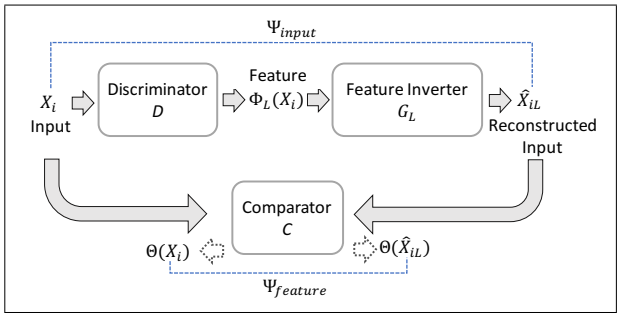

Figure 1: Functional block diagram of our feature inversion method. The method inverts a feature $\Phi_L(\mathbf{x}_i)$ from a layer L by training a feature inverter G_L that jointly minimises the input space loss Ψ_{input} and feature space loss $\Psi_{feature}$. Φ_L and Θ are the representation functions of a discriminator D and comparator C, respectively.

sis methods as it neither limits the analysis only to shallow layers nor puts any restriction on the depth of a model.

We demonstrate our method for a state-of-the-art singing voice detection (SVD) model that classifies an input mel spectrogram excerpt into two classes: 'vocal' and 'non-vocal' [37]. We first train up-convolutional neural networks [7], one per layer of the SVD model, to invert the features generated by it. We then quantify the performance of the inversion models (we call them "feature inverters") by calculating the normalised reconstruction error (NRE) that [23] defines as the normalised Euclidean distance between an input and its inverted representation. The results demonstrate that NRE is largest for a feature inverter that inverts the deepest layer (the last fully connected layer) in the SVD model and decreases for feature inverters that invert features from shallow layers. Finally, we qualitatively analyse the inverted features for both classes (vocal and non-vocal) to understand the preserved input content at each layer of the SVD model. Similarly, we analyse the inverted features for inputs selected from different datasets to test whether the conclusions from one dataset generalise to the other. The experimental code and results are available online. [1]

2. FEATURE INVERSION

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.

2.1 Prior Work

Mahendran et al. [23] and Dosovitskiy et al. [6] apply feature inversion to analyse the global behaviour of convolutional neural networks (CNNs). Mahendran et al. [23, 24] invert the features from AlexNet [18] (a CNN for image recognition). Their work demonstrates 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 demonstrate 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 highlights the invariances captured by the AlexNet layers. For example, the inverted representations from the deepest layer in the model (a fully connected layer) depict an object at different locations, orientations and scales.

The method introduced by Mahendran et al. [23] generates an inverted representation $\mathbf{x}^*_{iL} \in \mathbb{R}^n$ from an L^{th} layer feature by iteratively minimising the feature space loss between an input image $\mathbf{x}_i \in \mathbb{R}^n$ and an intermediate representation $\mathbf{x}'_{iL} \in \mathbb{R}^n$. The method starts with a randomly sampled $\mathbf{x}'_{iL}$ and in each iteration updates it by calculating the gradient of the loss function at $\mathbf{x}'_{iL}$. Formally, given a CNN with representation function $\Phi_L : \mathbb{R}^n \rightarrow \mathbb{R}^d$ that maps an n-dimensional input to a d-dimensional feature $\Phi_L(\mathbf{x}_i)$ at a layer L, the method inverts $\Phi_L(\mathbf{x}_i)$ by solving

$$\mathbf{x}^*_{iL} = \arg\min_{\mathbf{x}'_{iL}} \|\Phi_L(\mathbf{x}'_{iL}) - \Phi_L(\mathbf{x}_i)\|^2 + \alpha f(\mathbf{x}'_{iL}) \quad (1)$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a regularisation function (a natural image prior) that limits the search to realistic images and α is a scaling constant. Regularisation is needed since an unrestricted search may output fooling examples [30] that cause high activations to a neuron (or a layer), but do not possess features found in natural images. The method by Mahendran et al. [23, 24] has two major limitations: (1) hand-crafting a prior is challenging as for some inputs (e.g., images, audio) defining the constituents of a real input is difficult; and (2) the method needs to solve Eq. 1 for every new feature it needs to invert.

The feature inversion method proposed by Dosovitskiy et al. [6] tackles both the above issues and demonstrates visually improved reconstructions even for the fully connected layers of AlexNet. The method trains another neural network, an up-convolutional neural network (feature inverter) [7], to invert the features of a DNN. 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, 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.

2.2 Our Method

Fig. 1 provides an overview of our method. We use the approach of Dosovitskiy et al. [6], but modify its loss

[1] https://github.com/saum25/ISMIR-2018

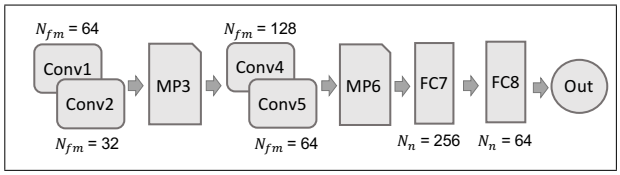

Figure 2: The architecture of the singing voice detection model proposed by Schlüter et al. [37]. N_{fm} denotes the number of feature maps in the output of a convolutional layer. N_n denotes the number of neurons in a fully connected layer. Conv: convolutional layer, MP: max-pooling layer, FC: fully connected layer, Out: output layer.

function to reduce the effect of input averaging. Recent works [5, 14] demonstrate that minimising loss in the perceptual space helps to reduce the over-smoothness problem for image generation models. We extend this idea to machine listening. Thus, in addition to the input space loss Ψ_{input}, our method also calculates the feature space loss $\Psi_{feature}$. We define total loss Ψ as:

$$\Psi = \lambda_{input}\Psi_{input} + \lambda_{feature}\Psi_{feature} \tag{2}$$

where λ_{input} and $\lambda_{feature}$ weight the losses of the input space and feature space. Thus, our method trains a feature inverter G_L to invert a feature $\Phi_L(\mathbf{x}_i)$ by jointly minimising the input space and feature space losses. To evaluate $\Psi_{feature}$, we use the approach from [5] where the authors 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 pre-trained discriminative model that may or may not be of the same depth as the discriminator D (the model whose features we are inverting). We can even use D as a comparator by extracting feature vectors at a layer of D (e.g., Dosovitskiy et al. [5] 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, our 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 $\Theta : \mathbb{R}^n \rightarrow \mathbb{R}^{d'}$, we define $\Psi_{feature}$ 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, we define 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

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_i (\|\mathbf{x}_i - G_L\{\Phi_L(\mathbf{x}_i); \mathbf{w}\}\|^2 + \|\Theta(\mathbf{x}_i) - \Theta(G_L\{\Phi_L(\mathbf{x}_i); \mathbf{w}\})\|^2) + \beta\|\mathbf{w}\|^2 \tag{3}$$

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

$$\hat{\mathbf{x}}_{iL} = G_L(\Phi_L(\mathbf{x}_i); \mathbf{w}^*) \tag{4}$$

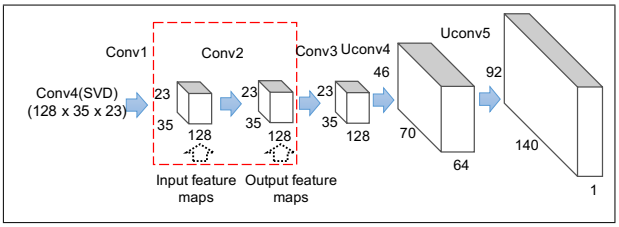

Figure 3: Feature inverter architecture for the Conv4 layer of the SVD model. The highlighted components represent the 'Conv2' convolutional layer and its input and output feature maps. Uconv: up-convolutional layer, Conv: convolutional layer.

3. INVERTING THE FEATURES OF A DEEP SINGING VOICE DETECTOR

We now demonstrate our feature inversion method from Section 2 for a state-of-the-art SVD model [37]. We first introduce the SVD model and then explain the architectures and training details of our feature inverters. Finally, we evaluate the performance of the feature inverters on two SVD datasets.

3.1 The Deep Singing Voice Detection Model

Singing voice detection is an audio segmentation problem where the task is to classify an input audio frame (excerpt) into one of the two categories: singing voice with or without instrumental music ('vocal') or instrumental music without singing voice ('non-vocal'). There exist several methods for singing voice detection. Some use handcrafted features to train shallow classifiers [20, 22, 34], while others jointly optimise the feature extraction and classification steps using deep learning [19, 21, 37].

Schlüter et al. [37] train an SVD model using a CNN and seven data augmentation techniques. Their model achieves state-of-the-art performance on public benchmark datasets. [2] Fig. 2 depicts the architecture of their 8-layered SVD model. Each convolutional layer performs convolution using 3×3 filters with 1×1 stride and no zero padding. The two max-pooling layers perform pooling with 3×3 stride and no zero padding. The input to the model is a mel spectrogram of about 1.6sec (115 frames). The model was trained on the Jamendo dataset [34]. Jamendo is a dataset of pop music songs and it consists of non-overlapping training, validation and test subsets with 61, 16 and 16 audio files, respectively. The model uses the auxiliary data (57 frames on each sides of the centre frame) as context to classify the centre frame in an input.

3.2 Feature Inverter Architectures

We train up-convolutional neural networks to invert the features generated by the above SVD model. We design

[2] https://github.com/f0k/ismir2015

Layer	Input Shape	Units	Output Shape
FC1	256×1	256	256×1
Reshape	256×1	-	$16 \times 4 \times 4$
Uconv2	$16 \times 4 \times 4$	64	$64 \times 8 \times 8$
Conv3	$64 \times 8 \times 8$	64	$64 \times 8 \times 8$
Uconv4	$64 \times 8 \times 8$	32	$32 \times 16 \times 16$
Conv5	$32 \times 16 \times 16$	32	$32 \times 16 \times 16$
Uconv6	$32 \times 16 \times 16$	16	$16 \times 32 \times 32$
Conv7	$16 \times 32 \times 32$	16	$16 \times 32 \times 32$
Uconv8	$16 \times 32 \times 32$	8	$8 \times 64 \times 64$
Conv9	$8 \times 64 \times 64$	8	$8 \times 64 \times 64$
Uconv10	$8 \times 64 \times 64$	1	$1 \times 128 \times 128$

Table 1: Feature inverter architecture to invert the FC7 layer of the SVD model. Input and output shape dimensions are ordered as the number of channels $\times$ time $\times$ frequency. Uconv: up-convolutional layer, Conv: convolutional layer, FC: fully connected layer. Units refer to the number of neurons in a fully connected layer or the number of filters in a convolutional layer.

Inv-idx	Inv-inp	Inv-depth	Inv-nconv
FC8	64×1	11	4
FC7	256×1	10	4
MP6	$64 \times 11 \times 7$	5	1
Conv5	$64 \times 33 \times 21$	5	3
Conv4	$128 \times 35 \times 23$	5	3
MP3	$32 \times 37 \times 25$	6	4
Conv2	$32 \times 111 \times 76$	4	4
Conv1	$64 \times 113 \times 78$	2	2

Table 2: Architectural overview of the feature inverters for all the layers in the SVD model. Inv-idx: SVD layer a feature inverter inverts, Inv-inp: input to a feature inverter (number of channels $\times$ time $\times$ frequency), Inv-depth: number of layers in a feature inverter, Inv-nconv: number of convolutional layers in a feature inverter. Conv: convolutional layer, FC: fully connected layer and MP: max-pooling layer.

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 the SVD model. [3] The architecture of inversion models in [6] inspires the design of our feature inverters, but we adapt the architectures to suit the SVD model. A majority of feature inverters need to perform the upsampling (unpooling) operation that is an approximate inverse of the max-pooling operation done in the SVD model. In order to perform unpooling and convolution in a single step, we use up-convolutional layers (Uconv) with 4×4 filters and 2×2 stride. This configuration of Uconv layers upsamples an input feature map by 2 [7] . The number of such layers depends on the dimensionality of the layer we are inverting. For example, the feature inverter to invert the 256-dimensional FC7 layer uses 5 Uconv layers (Table 1), while the feature inverter to invert the Conv4 layer uses two Uconv layers (Fig. 3). The feature inverters for the Conv1 and Conv2 layers in the SVD model do not use Uconv layers as for them the model generates features without using the max-pooling layer.

We increase the capacity of the feature inverters by adding convolutional layers; either after every up-convolutional layer (for inverting an FC layer) or before the first up-convolutional layer (for inverting a Conv or MP layer). We empirically decide the number of convolutional layers for each feature inverter. The convolutional layers perform convolution using 3×3 filters with 1×1 stride and improve the visual appearance of the reconstructions [8]. Table 2 provides details about the depth and the number of Conv layers in each feature inverter. All the layers use exponential linear unit (ELU) non-linearity given by $y(x) = x$ if $x > 0$, otherwise $e^x - 1$ [2]. The network uses batch normalisation layers [13] to make sure the input to each layer follows a standard normal distribution. 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.

3.3 Training of the Feature Inverters

We train one feature inverter per layer of the SVD model. We train a feature inverter using mel spectrogram excerpts of about 1.6 sec that we extract from the Jamendo training dataset. We show one such sample in Fig. 4. We generate excerpts with a hop size of 10 frames (140 ms). Thus, we train each feature inverter using a data set of about $100k$ features. We do not use any data augmentation techniques. In order to prevent overfitting, we run the optimisation to a fixed number of weight updates (30 epochs) and select a feature inverter giving the lowest loss on the validation subset. We use the Conv5 layer of the SVD model as the comparator, i.e., we encode the mel spectrogram and the inverted representation using Conv 5. We initialise the feature inverter weights using the He normal initialisation method [12]. In each iteration, for a mini-batch of 32 randomly selected excerpts, the training objective jointly minimises the feature and input space losses and updates the change in parameters using ADAM [17]. We set the scaling factors λ_{input} and $\lambda_{feature} = 1$ (Eq. 2). We start training with an initial learning rate of 0.001 and decay it by 0.5 when the training loss does not change for 2 consecutive epochs. The training procedure performs regularisation using L_2 weight decay and sets $\beta = 1e - 4$ (Eq. 3).

3.4 Quantitative Evaluation of the Feature Inverters

We train eight feature inverters using the Jamendo training dataset and the architectures and training methodology discussed above. We evaluate the performance of each feature inverter on an evaluation set of 128 mel spectrogram excerpts. We build the evaluation set by randomly selecting

[3] "inverting a layer" is another way to refer to the inversion of the features generated by a layer.

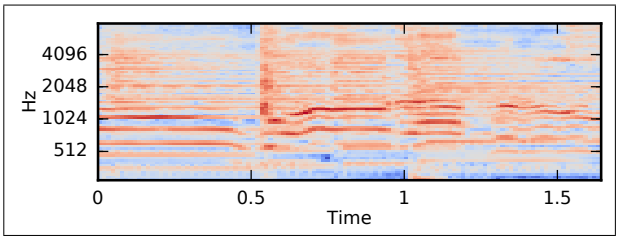

Figure 4: Sample mel spectrogram excerpt from the Jamendo dataset. This excerpt belongs to the vocal class.

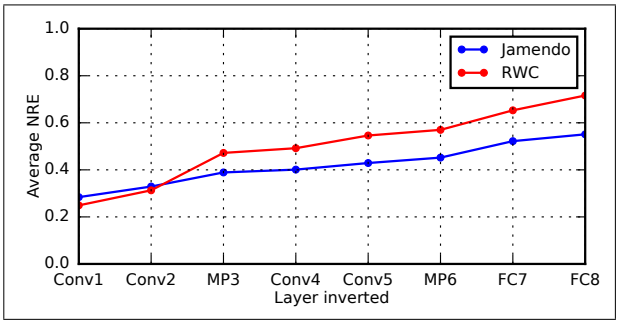

Figure 5: Performance evaluation of the feature inverters. The plot depicts the change in average normalised reconstruction error (NRE) as a feature inverter inverts different layers in the SVD model.

8 excerpts from each of the 16 audio files in the Jamendo test dataset. We quantify the performance of feature inverters by calculating the average normalised reconstruction error (NRE) for each feature inverter on the evaluation dataset. [23] defines NRE as:

$$NRE = \|\mathbf{x}_i - \hat{\mathbf{x}}_{iL}\|/N_c \quad (5)$$

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

We also evaluate the feature inverters on the RWC dataset [25] to understand whether the results of the quantitative evaluation on Jamendo extend to the RWC dataset. The RWC dataset for singing voice detection is a public benchmark dataset that contains a collection of 100 pop music songs, but unlike Jamendo, there is no partitioning into separate subsets. Thus, to evaluate our models we first build an RWC test dataset by randomly selecting 20 audio files from a set of 100 and use them to build an evaluation dataset of 160 randomly selected excerpts (8 excerpts per audio files). Moreover, in order to evaluate the feature inverters on a larger evaluation dataset, we randomly sample 10 different evaluation sets, calculate the average NRE for each and later take an average. Thus, effectively we evaluate our feature inverters on an evaluation dataset of size 1280 (Jamendo) and 1600 (RWC) excerpts.

Fig. 5 shows the results of the evaluation. For both datasets, the reconstruction error is largest for the deepest layer in the SVD model (FC8) and 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, making it easier to invert them. For instance, the dimensionality reduction of features from MP6 to FC7 is about 19 times, compressing a 4928-dimensional feature to 256 dimensions. Similarly, we see 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 also depict that the feature inverters have larger reconstruction error on the RWC dataset at all but two layers. This is expected since both the discriminator (the SVD model) and the feature inverters are trained on the Jamendo dataset. One possible explanation for the comparable average NRE of the Conv1 and Conv2 layers is that these shallow layers of the model are learning generalisable features [47]. This becomes less so at deeper layers, where features are likely tuned to specific traits of the training data.

We also compare the performance of the feature inversion method we use in this work (we call it 'M_{joint}') with a baseline method (we call it 'M_{input}') that trains feature inverters using image loss only. We train and test the feature inverters of M_{input} on the Jamendo dataset. We find that across all the layers of the SVD model, the average NRE of the feature inverters using M_{input} is either similar or slightly lower than for those using M_{joint}. Such a behaviour is predictable as M_{input} aims to only minimise the input reconstruction loss, while M_{joint} aims to jointly optimise both the loss functions, which may or may not result in lower NRE [5]. The benefit of using M_{joint} comes from the generation of inverted representations that are perceptually closer to an input, a property that is challenging for M_{input} to achieve.

4. QUALITATIVE ANALYSIS OF THE INVERTED FEATURES

Fig. 6 shows visualisations for each layer of the SVD model. We generate these visualisations by selecting four inputs, two from each dataset (Jamendo and RWC), in which one belongs to each of the two classes (vocal and non-vocal). We then use the feature inverters to invert the features extracted by the SVD model from each input. The results provide some insights into the model behaviour. For example, reconstructions from the FC8 layer suggest that FC8 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 cases (in addition to these four inputs), we found a clear demarcation between the vocal and the non-vocal class visualisations from this layer. We find that for the vocal class, energy appears in higher frequencies while for the non-vocal class energy appears in lower frequencies. These visualisations suggest that the SVD model learns a class-decision function in this layer.

Similarly, reconstructions from the FC7 layer suggest that the layer preserves some harmonic content and ap-

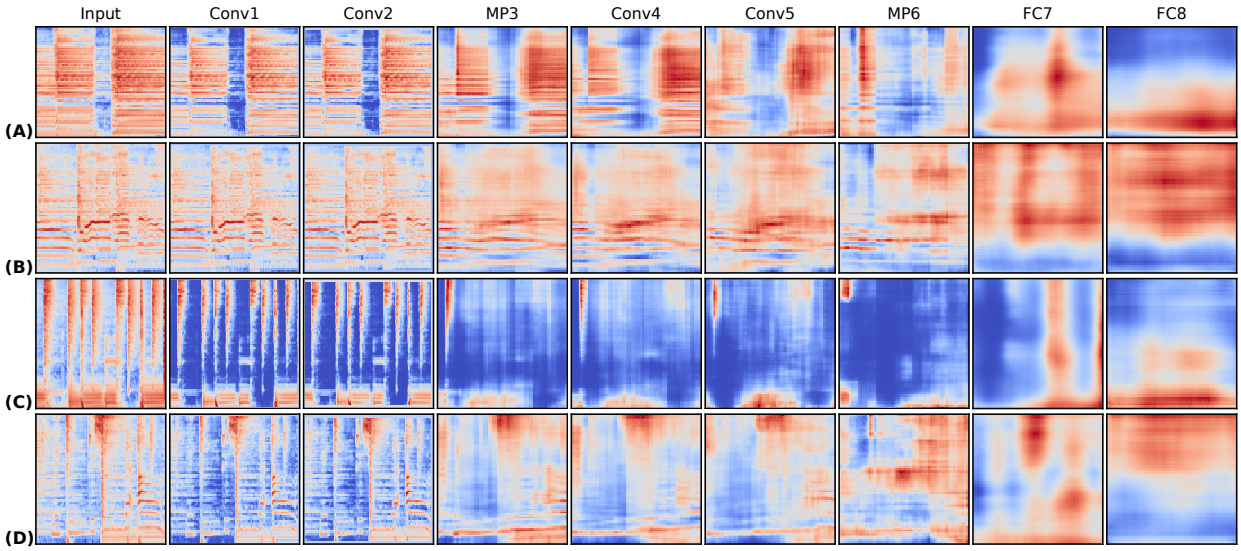

Figure 6: Feature inversion from successive layers of the SVD model. Each row corresponds to one input excerpt: (A), (B) are respectively non-vocal and vocal excerpts from "03 - Say me Good Bye.mp3" in the Jamendo test dataset. Similarly, (C) and (D) are respectively non-vocal and vocal excerpts from "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 then inverted representations from successive SVD model layers (as labelled). The visualisations highlight how the model ignores the input content as it forms higher-level representations. Inversions of shallow layers resemble the input, but the reconstruction quality reduces for deeper layers. Conv: convolutional layer, MP: max-pooling layer, FC: fully connected layer.

proximate onset locations of the inputs. But, there are some deviations from this behaviour. For instance, in Fig. 6 row B, the harmonic structures are less evident. Similarly, for the input in row C, the feature inverter at FC7 is unable to reconstruct all the harmonic and temporal content present in the input. This may be due to the fact that we do not train the feature inverters on RWC, thus the reconstruction error is higher for this input, resulting in poor reconstruction.

We find that reconstructions from the deepest convolutional layer of the model contain more information than those from the two fully connected layers. For both inputs from Jamendo (Fig 6A-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 the quantitative results of model inversion for Conv5 and FC7 layers, where we show that the average NRE is about 18% less for Conv5. The visualisations for the RWC excerpts (Fig. 6C-D) report 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 harmonics 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 2 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.

5. CONCLUSION AND FUTURE WORK

In this work, we applied a model analysis method called feature inversion to a state-of-the-art singing voice detection model. Feature inversion helped to understand the global behaviour of the SVD model by visualising the information preserved by any layer in the model. We trained up-convolutional neural networks to invert the features of the model. We quantitatively analysed the feature inverters for each layer in the model to understand the change in input reconstruction error across different layers. We found that the average NRE changes by about 15% for Jamendo between the MP6 and FC7 layers due to high dimensionality reduction. Moreover, we qualitatively visualised the inverted representations to understand the input content preserved by any layer in the model. We found that the deepest fully connected layer does not retain any of the temporal or harmonic structures present in an input. We also found that for a large number of inputs this layer seems to learn a decision function that depends on the class associated with an input. Qualitative analysis of other layers revealed that the FC7 layer preserves some harmonic and temporal information of an input while the reconstructions from the Conv5 layer are visually similar to the input.

In our future work, we plan to improve the loss function by adding adversarial loss [5] that helps to generate realistic inverted representations that are close to one of the classes. This facilitates sonification of inverted representations, giving more insights into the model behaviour. Moreover, we plan to extend the analysis by applying feature inversion to different architectures of the SVD model and also to different machine listening tasks.

6. ACKNOWLEDGEMENTS

We would like to thank Jan Schlüter for discussions and sharing his implementation of the SVD model. We also thank the anonymous reviewers for their valuable comments and suggestions.

7. REFERENCES

[1] K. Choi, G. Fazekas, and M. B. Sandler. Explaining Deep Convolutional Neural Networks on Music Classification. *arXiv e-prints*, arXiv:1607.02444, 2016.

[2] DA. Clevert, T. Unterthiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *ArXiv e-prints*, arXiv:1511.07289, 2015.

[3] S. Dieleman and B. Schrauwen. End-to-End Learning for Music Audio. In *Proc. ICASSP*, 2014.

[4] F. Doshi-Velez and B. Kim. Towards a Rigorous Science of Interpretable Machine Learning. *arXiv e-prints*, arXiv:1702.08608, 2017.

[5] A. Dosovitskiy and T. Brox. Generating Images with Perceptual Similarity Metrics Based on Deep Networks. In *Proc. NIPS*, 2016.

[6] A. Dosovitskiy and T. Brox. Inverting Visual Representations with Convolutional Networks. In *Proc. CVPR*, 2016.

[7] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to Generate Chairs with Convolutional Neural Networks. In *Proc. CVPR*, 2015.

[8] A. Dosovitskiy, J. T. Springenberg, M. Tatarchenko, and T. Brox. Learning to Generate Chairs, Tables and Cars with Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[9] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualising Higher-Layer Features of a Deep Network. Technical Report 1341, University of Montreal, June 2009.

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

[11] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adverserial Examples. In *Proc. ICLR*, 2015.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proc. ICCV*, 2015.

[13] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proc. ICML*, 2015.

[14] J. Snell and K. Ridgeway and R. Liao and B.D. Roads and M. C. Mozer and R. S. Zemel. Learning to Generate Images with Perceptual Similarity Metrics. In *Proc. ICIP*, 2017.

[15] C. Kereliuk, Bob L. Sturm, and J. Larsen. Deep Learning, Audio Adversaries, and Music Content Analysis. In *Proc. WASPAA*, 2015.

[16] PJ. Kindermans, S. Hooker, J. Adebayo, M. Alber, K. T. Schütt, S. Dähne, D. Erhan, and B. Kim. The (Un)reliability of Saliency Methods. In *Proc. NIPS Workshop*, 2017.

[17] D. Kingma and B. Jimmy. Adam: A Method for Stochastic Optimization. In *Proc. ICLR*, 2015.

[18] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Proc. NIPS*, 2012.

[19] S. Leglaive, R. Hennequin, and R. Badeau. Singing Voice Detection with Deep Recurrent Neural Networks. In *Proc. ICASSP*, 2015.

[20] B. Lehner, R. Sonnleitner, and G. Widmer. Towards Light-Weight, Real-Time-Capable Singing Voice Detection. In *Proc. ISMIR*, 2013.

[21] B. Lehner, G. Widmer, and S. Bock. A Low-Latency, Real-Time-Capable, Singing Voice Detection Method with LSTM Recurrent Neural Networks. In *Proc. EUSIPCO*, 2015.

[22] B. Lehner, G. Widmer, and R. Sonnleitner. On the Reduction of False Positives in Singing Voice Detection. In *Proc. ICASSP*, 2014.

[23] A. Mahendran and A. Vedaldi. Understanding Deep Image Representations by Inverting Them. In *Proc. CVPR*, 2015.

[24] A. Mahendran and A. Vedaldi. Visualizing Deep Convolutional Neural Networks Using Natural Pre-images. *International Journal of Computer Vision*, pages 1–23, 2016.

[25] M. Mauch, H. Fujihara, K. Yoshii, and M. Goto. Timbre and Melody Features for the Recognition of Vocal Activity and Instrumental Solos in Polyphonic Music. In *Proc. ISMIR*, 2011.

[26] S. Mishra, B. L. Sturm, and S. Dixon. Local Interpretable Model-Agnostic Explanations for Music Content Analysis. In *Proc. ISMIR*, 2017.

[27] G. Montavon, W. Samek, and KR. Müller. Methods for Interpreting and Understanding Deep Neural Networks. *Digital Signal Processing*, 73(Supplement C):1–15, 2018.

[28] A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going Deeper into Neural Networks . "`https://research.googleblog.com/2015/06/`

inceptionism-going-deeper-into-neural.html", 2015.

[29] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the Preferred Inputs for Neurons in Neural Networks Via Deep Generator Networks. In *Proc. NIPS*, 2016.

[30] A. Nguyen, J.Yosinski, and J. Clune. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In *Proc. CVPR*, 2015.

[31] C. Olah, A. Mordvintsev, and L. Schubert. Feature Visualization. *Distill*, 2017.

[32] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical Black-Box Attacks against Machine Learning. In *Proc. ACM ASIACCS*, 2017.

[33] R. Vidal and J. Bruna and R. Giryes and S. Soatto. Mathematics of Deep Learning. *ArXiv e-prints*, arXiv:1712.04741, 2017.

[34] M. Ramona, G. Richard, and B. David. Vocal Detection in Music Using Support Vector Machines. In *Proc. ICASSP*, pages 1885–1888, 2008.

[35] M. T. Ribeiro, S. Singh, and C. Guestrin. Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proc. KDD*, 2016.

[36] J. Schlüter and S. Böck. Improved Musical Onset Detection with Convolutional Neural Networks. In *Proc. ICASSP*, 2014.

[37] J. Schlüter and T. Grill. Exploring Data Augmentation for Improved Singing Voice Detection with Neural Networks. In *Proc. ISMIR*, 2015.

[38] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual Explanations from Deep Networks Via Gradient-Based Localization. In *Proc. ICCV*, 2017.

[39] A. Shrikumar, P. Greenside, and A. Kundaje. Learning Important Features Through Propagating Activation Differences. In *Proc. ICML*, pages 3145–3153, 2017.

[40] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. In *Proc. ICLR*, 2014.

[41] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. Smoothgrad: Removing Noise by Adding Noise. In *Proc. ICML Workshop on Visualisation for Deep Learning*, 2017.

[42] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for Simplicity: The All Convolutional Net. In *Proc. ICLR Workshop*, 2015.

[43] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic Attribution for Deep Networks. In *Proc. ICML*, 2017.

[44] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing Properties of Neural Networks. In *Proc. ICLR*, 2014.

[45] A. Weller. Challenges for Transparency. In *Proc. ICML Workshop on Human Interpretability in Machine Learning*, 2017.

[46] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding Neural Networks Through Deep Visualization. In *Proc. ICML Deep Learning Workshop*, June 2015.

[47] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In *Proc. ECCV*, 2014.

[48] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding Deep Learning Requires Rethinking Generalization. In *Proc. ICLR*, 2017.