Deep Learning and Music Adversaries
STURM, BLT; Kereliuk, C; Larsen, J

© 2015, IEEE

For additional information about this publication click this link.
http://qmro.qmul.ac.uk/xmlui/handle/123456789/11150

Information about this research object was correct at the time of download; we occasionally make corrections to records, please therefore check the published record when citing. For more information contact scholarlycommunications@qmul.ac.uk
Deep Learning and Music Adversaries

Corey Kereliuk, Member, IEEE, Bob L. Sturm, Member, IEEE, Jan Larsen Member, IEEE

Abstract—An adversary is essentially an algorithm intent on making a classification system perform in some particular way given an input, e.g., increase the probability of a false negative. Recent work builds adversaries for deep learning systems applied to image object recognition, which exploits the parameters of the system to find the minimal perturbation of the input image such that the network misclassifies it with high confidence. We adapt this approach to construct and deploy an adversary of deep learning systems applied to music content analysis. In our case, however, the input to the systems is magnitude spectral frames, which requires special care in order to produce valid input audio signals from network-derived perturbations. For two different train-test partitionings of two benchmark datasets, and two different deep architectures, we find that this adversary is very effective in defeating the resulting systems. We find the convolutional networks are more robust, however, compared with input audio signals from network-derived perturbations. For two case, however, the input to the systems is magnitude spectral frames, which makes a classification system perform in some particular way, e.g., increase the probability of a false negative. Recent work builds adversaries for deep learning systems applied to image object recognition, which exploits the parameters of the system to find the minimal perturbation of the input image such that the network misclassifies it with high confidence. We adapt this approach to construct and deploy an adversary of deep learning systems applied to music content analysis. In our case, however, the input to the systems is magnitude spectral frames, which requires special care in order to produce valid input audio signals from network-derived perturbations. For two different train-test partitionings of two benchmark datasets, and two different deep architectures, we find that this adversary is very effective in defeating the resulting systems. We find the convolutional networks are more robust, however, compared with input audio signals from network-derived perturbations. For two case, however, the input to the systems is magnitude spectral frames, which requires special care in order to produce valid input audio signals from network-derived perturbations. For two different train-test partitionings of two benchmark datasets, and two different deep architectures, we find that this adversary is very effective in defeating the resulting systems. We find the convolutional networks are more robust, however, compared with input audio signals from network-derived perturbations.
II. DEEP LEARNING FOR MUSIC CONTENT ANALYSIS

We first provide an overview of research in applying deep learning approaches to music content analysis. We then discuss two different architectures, train two music content analysis systems, and test them in two benchmark MIR datasets. These systems are the subjects of our experiments in Section V.

A. Overview

Artificial neural networks have been applied to many music content analysis problems, for instance, fingerprinting, genre recognition, emotion recognition, artist recognition, and even composition. Advances in training have enabled the creation of more advanced and deeper architectures. Deng and Yu (Chapter 7) provide a review of successful applications of deep learning to the analysis of audio, highlighting in particular its significant contributions to speech recognition in conversational settings. Humphrey et al. [31] provide a review for applications to music in particular, and motivate the capacity of deep architectures to automatically learn hierarchical relationships in accordance with the hierarchical nature of music: “pitch and loudness combine over time to form chords, melodies and rhythms.” They argue that this is key for moving beyond the reliance on “shallow” and hand-designed features that were designed for different tasks.

Lee et al. [34] are perhaps the first to apply deep learning to music content analysis, specifically genre and artist recognition. They train a convolutional deep belief network (CDBN) with two hidden layers in an unsupervised manner to train/test task using a standard machine learning approach. A table of their experimental results, using some portion of the dataset ISMIR2004, shows higher accuracies for their deep learned features compared to those for standard MFCCs. For genre recognition, Li et al. [35] use convolutional deep neural networks (CDNN) with three hidden layers, into which they input a sequence of 193-dimensional MFCC feature vectors. The architecture of their CDNN is such that the first hidden layer considers data from 127 ms duration, and the last hidden layer is capable of summarising events over a 2.2 s duration. van den Oord et al. [57] apply CDNN to mel-frequency spectrograms for automatic music content analysis.

For genre recognition and more general descriptors, Hamel and Eck [28] train a DNN with three hidden layers of 50 units each, taking as input 513 discrete Fourier transform (DFT) magnitudes computed from a single 46 ms audio frame. They use a train/valid/test partition of the benchmark music genre dataset GTzan [49], [55]. They also explore “aggregated” features, which are the mean and variance in each dimension of activations over 5 second durations. They find in the test set, and for both short-term and aggregated features, that SVM classifiers trained with features built from hidden layer activations reproduce more ground truth than an SVM classifier trained with features built from MFCCs. They report an accuracy of over 0.84 for features that aggregate activations of all three hidden layers. Sigetia and Dixon [44] explore modifications to the system in [28], in particular using different combinations of architectures, training procedures, and regularisation. They use the activations of their trained DNN as features for a train/test task using a random forest classifier. They report an accuracy of about 0.83 using features aggregating activations of all hidden layers of 500 units each. For genre recognition, Yang et al. [63] combine 263-dimensional modulation features with a DBN. For music rhythm classification, Pikrakis [41] employs a DBN, which we studied further in [51].

Dielemans et al. [18] build and apply CDBN to music key detection, artist recognition, and genre recognition. There are three major differences with respect to the work above [28], [34], [35], [44], [63]. First, Dielemans et al. employ 24-dimensional input features computed by averaging short-time chroma and timbre features over the time scales of single musical beats. Second, they employ expert musical knowl-
edge to guide decisions about the architecture of the system. Finally, they use the output posteriors of their system for classification, instead of using the hidden layer activations as features for a separate classifier. Their experiments in a portion of the “million song dataset” show large differences in classification accuracies between their systems and a naive Bayesian classifier using the same input features. In a unique direction for audio, Dieleman and Schrauwen [19] explore “end-to-end” learning, where a CDNN is trained with input of about 3 s of raw audio samples for a music content analysis task (autotagging). They find that the lowest layer of the trained CDNN appears to learn some filters that are frequency selective. They evaluate this system for a multilabel problem.


C. Deep learning with two music genre benchmarks

We now build DNNs and CDNNs using two music genre benchmarks: GTZAN [49, 55] and the Latin Music Database (LMD) [45]. GTZAN consists of 100 30-second music recording excerpts in each of ten categories, and is the most-used public dataset in MIR research [50]. LMD is a private dataset, consisting of 3,229 full-length music track recordings non-uniformly distributed among ten categories, and has been used in the annual MIREX audio latin music genre classification evaluation campaign since 2008. We use the first 30 seconds of each track in LMD.

We build several DNNs and CDNNs using different partitionings of these datasets. One partitioning of GTZAN we create by randomly selecting 500/250/250 excerpts for training/validation/testing. The other partitioning of GTZAN is “fault-filtered,” which we construct by hand to include 443/197/290 excerpts. This involves removing 70 files including exact replicas, recording replicas, and distorted files [49], and then dividing the excerpts such that no artist is repeated across the training, validation, and test partitions. We partition LMD in two ways: 1) partitioning by 60/20/20% sampling

\[\text{http://www.music-ir.org/mirex/wiki/MIREX_HOME} \]
in each class; 2) a hand-constructed artist-filtered partitioning containing approximately the same division of excerpts in each class. We retain all 213 replicas in LMD.[49]

The input to our systems is derived from the short-time Fourier transform (STFT) of a sampled audio signal \( x \):\(^{(1)} \)

\[
F(x)[m, u] = \sum_{l=0}^{L-1} w[l] |x[l - u H] e^{-j 2 \pi m l / L} \tag{1}
\]

where the parameter \( L \) defines both the window length and the number of frequency bins. We define \( w \) as a Hann window of length \( L = 1024 \), which corresponds to a duration of 46ms for recordings sampled at 22050 Hz. The window is hopped along \( x \) with a stride of \( H = 512 \) samples (adjacent windows overlap by 50%).

Since audio signals can be of any duration, we define the input to our systems as a sequence \( X = (X_n)_{n=0}^{N-1} \), where the sequence length depends on the input audio’s duration. We define the \( n \)th element of the input sequence \( X \) to be

\[
X_n = \left( F(x)[m, u] : m \in [0, 512], u \in [nT, (n+1)T] \right) \tag{2}
\]

where \( T = 1 \) for each DNN and \( T = 100 \) for each CDNN. Thus, when \( T = 1 \), \( X \) is a sequence of 513 \( \times \) 1 vectors; when \( T = 100 \), \( X \) is a sequence of 513 \( \times \) 100 matrices.

A (C)DNN processes each element in this sequence independently, outputting a sequence \( P = (P_n)_{n=0}^{N-1} \) from the final (softmax) layer. The output vector \( P_n \in \{0, 1\}^K \), ||\( P_n ||_1 = 1 \), is the posterior distribution of labels assigned to the \( n \)th element in the input sequence by the network. Therefore, we may write \( P_n(i)X_n, \Theta) \Rightarrow P_n(i) \in [0, 1] \) where \( \Theta \) represents the trainable network parameters, i.e., the set of weights and biases. We define the confidence of a (C)DNN in a particular label \( k \in \{1, \ldots, K\} \) for an input sequence \( X \) as the sum of all posteriors, i.e.,

\[
R(k|X, \Theta) = \frac{1}{N} \sum_{n=0}^{N-1} P_n(k|X_n, \Theta). \tag{3}
\]

We apply a label to an input sequence \( X \) as the one maximising the confidence

\[
y(X, \Theta) = \arg \max_{k \in \{1, \ldots, K\}} R(k|X, \Theta). \tag{4}
\]

Paralleling the work in [44], we build DNNs with 3 fully connected hidden layers, and either 50 or 500 units per layer. Our CDNN has two convolutional layers (accompanied by max pooling layers) followed by a fully connected hidden layer with 50 units. Figure [1] illustrates the architecture of our CDNN. Its first convolutional layer contains 32 filters, each arranged in a rectangular 400 \( \times \) 4 grid. We choose this long rectangular shape instead of the small square patches typically used when training on images based on our knowledge that many sounds exhibit strong harmonic structures that span a large portion of the audible spectrum. The second convolutional layer contains 32 filters, each connected in an 8 \( \times \) 8 pattern. Our two pooling neighborhoods are 4 \( \times \) 4 and have strides of 2 \( \times \) 2. All of our deep learning systems use rectified linear units (ReLUs), and have a softmax unit in the final layer. As is typical, we standardise the (C)DNN inputs by subtracting the training set mean and dividing by the standard deviation in each of the input dimensions. We perform this with a linear layer above the input layer of each network. The raw inputs to the network are still \( X_n \).

Also paralleling [44], we build several music classification systems treating our DNN as a feature extractor. In this case, we construct a set of features by concatenating the activations from the DNN’s three hidden layers, and aggregating them over 5-second texture windows (hopped by 50%). The aggregation summarises the mean and standard deviation of the feature dimensions over the texture window and may be seen as a form of late-integration of temporal information. We use this new set of features to train a random forest (RF) classifier [29] with 500 trees. Thus, to classify a music audio recording \( x \) from its set of aggregated features, we use majority voting over all classifications, which is also used in [44].

### D. Preliminary evaluation

Figure [2] and Table [I] show the results of RF classification using the features produced by the DNN when trained on GTZAN with the two different partitioning strategies; and Fig. [3] shows those for the (C)DNNs we train and test in LMD. Across each partition strategy we see significant differences in performance. The mean recall in each class in Figure [2] on the fault-filtered partition is much lower than that on the random test partition — involving drops higher than 30 percentage points in most cases. Table [I] shows similar drops in performance that persist over the inclusion of drop-out regularisation. Such significant drops in performance from partitioning based on artists is not unusual, and has been studied before as a bias coming from the experimental design [23], [39], [49]. Partitioning a music genre recognition dataset along artist lines has been recommended to avoid this bias [23], [39], and is in fact used in several MIREX audio classification tasks [7]. Experiments using GTZAN with fault-filtering partitioning has not been used in many benchmark experiments with GTZAN because its artist information has only recently been made available [49].

### III. Adversaries in music content analysis

An adversary is an agent that tries to defeat a classification system in order to maximise its gain, e.g., SPAM detection. Dalvi et al. [16] pose this problem as a game between a classifier and adversary, and analyse the strategies involved for an adversary with complete knowledge of the classification system, and for a classifier to adapt to such an adversary. Szegedy et al. [54] propose using adversaries for testing the assumption that deep learning systems are “smooth classifiers,” i.e., stable in their classification to small perturbations around examples in the training data. They define an adversary of a classifier \( f: \mathbb{R}^m \rightarrow \{1, \ldots, K\} \) as an algorithm using complete knowledge of the classifier to perturb an observation \( x \in \mathbb{R}^m \) such that \( f(x + r) \neq f(x) \), where \( r \in \mathbb{R}^m \) is some

---

[1]: https://highnoongmt.wordpress.com/2014/02/08/faults_in_the_latin_music_database
of the partial derivative of the loss function used to train a particular network, evaluated with the requested class. They also find that the direction of perturbation is important, not necessarily its size. Hence, it seems adversarial examples of one model will likely fool other models because they occur in large volumes in high-dimensional spaces. This is also found by Gu and Rigazio [27].

As for Szegedy et al. [54], we are interested in the robustness of our deep learning music content analysis systems to an adversary. Do these systems suffer just as dramatically as the image content recognition systems in [24], [27], [54]? In other words, can we find imperceptible perturbations of audio recordings, yet make the systems produce any label with high confidence? If so, can we adapt the training of the systems such that they become more robust? In the next subsections, we define an adversary as an optimisation problem, but with care of the fact that the input to our deep learning systems are magnitude STFT [2]. We then present an approach to integrate adversaries into the training of our systems. We present our experimental results in Section [IV].

A. Adversaries for music audio

The explicit goal of our adversary is to perturb a music recording $x$ such that a system will confidently classify it with some class $y \in \{1, \ldots, K\}$. Specifically, we define the adversary as the constrained optimisation problem:

$$\hat{X}(y) = \arg\min_{Z \in C(X)} \sum_{n=0}^{N-1} L(Z_n, y(\Theta))$$  (6)

Fig. 2. Figure of merit (FoM, ×100) in GTZAN with two different partitions for random forest classification (majority vote) of DNN-based features (all layers) aggregated over 5 second windows (mean and standard deviations). Each DNN has 500 rectified linear units in each hidden layer. Columns represent the true class; rows denote labels chosen by system; the diagonal contains the per-class recall; the off-diagonal entries are confusions; the rightmost column is the precision; the bottom row is the F-score; and the last element along the diagonal is the mean recall (normalised classification accuracy).

<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Layer</th>
<th>ReLU</th>
<th>ReLU+Dropout</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1</td>
<td>76.80 (40.69)</td>
<td>80.40 (45.17)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>78.80 (45.17)</td>
<td>80.40 (43.10)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>79.60 (43.79)</td>
<td>78.80 (44.48)</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>80.40 (43.79)</td>
<td>80.00 (43.79)</td>
</tr>
<tr>
<td>500</td>
<td>1</td>
<td>68.40 (40.34)</td>
<td>75.60 (40.69)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>74.40 (40.69)</td>
<td>80.00 (50.34)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>77.60 (43.79)</td>
<td>79.20 (48.62)</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>76.00 (42.41)</td>
<td>81.20 (48.97)</td>
</tr>
</tbody>
</table>

TABLE I

Mean normalised classification accuracy (×100) in GTZAN for random forest classification of DNN-based features from layer shown aggregated over 5-second windows. Number outside brackets is from random partition in Fig 2(a), and that inside brackets is from fault-filtered partition in Fig 2(b).

...
where we define the feasible set of adversarial examples to input sequence $X$ as:

$$C(X) = \left\{ Z = (Z_n)_{n=0}^{N-1} : \sqrt{\sum_{n=0}^{N-1} \| Z_n - X_n \|_2^2} \leq N\epsilon(SNR) \right\}$$

with the parameter

$$\epsilon(SNR) = \frac{1}{\sqrt{N}} \sqrt{\sum_{n=0}^{N-1} \| X_n \|_2^2}$$

limiting the maximum acceptable perturbation caused by the adversary. The loss function in (6) is the cross-entropy loss function, $\mathcal{L}(X, y|\Theta) := -\log P_n(y|X_n, \Theta)$, which we use in training our (C)DNNs. Given the network parameters $\Theta$, this adversary can compute the derivative of this loss function by backpropagating derivatives through the network. This suggests that our adversary can accomplish its goal by searching for a new input sequence $\hat{X}$ via gradient descent on the loss function with any label $y$ that differs from the ground truth. This is the approach used by Szegedy et al. [5] in the context of image object recognition.

A local minimum of (6) can be found using projected gradient descent, initialised with the exemplar $X^{(0)} \leftarrow X$, and iterating

$$\hat{X}^{(k+1)} \leftarrow \mathcal{P}_C(\hat{X}^{(k)} + \mu \nabla \mathcal{L}(\hat{X}^{(k)}, y|\Theta))$$

where the scalar $\mu$ is the gradient descent step size, and
Algorithm 1 From exemplar sequence $X$ search for adversarial sequence $\hat{X}$ with maximal perturbation SNR in at most $k_{\text{max}}$ steps that makes a (C)DNN with parameters $\Theta$ apply label $y$ with confidence $R_{\text{min}}$.

1: parameters: $y$, SNR, $\mu$, $R_{\text{min}}, \Theta, k_{\text{max}}$
2: init: $X^{(0)} = X, k = 0$
3: repeat
4: $V \leftarrow X^{(k)} + \mu \nabla L(X^{(k)}, y(\Theta)) \{\text{Gradient step}\}$
5: $W \leftarrow \mathcal{P}_{\mathcal{C}}(\text{max}(0, V)) \{\text{Find valid sequence}\}$
6: $\nu \leftarrow \max(0, \frac{1}{N} \sum_{n=0}^{N-1} ||W - X_n||_2^2/\epsilon(\text{SNR}) - 1)$
7: $X^{(k+1)} \leftarrow (1 + \nu)^{-1}(W + \nu X) \{\text{SNR constraint}\}$
8: $k \leftarrow k + 1$
9: until $\frac{1}{N} \sum_{n=0}^{N-1} P_n(y|X^{(k+1), \Theta}) \geq R_{\text{min}}$ or $k = k_{\text{max}}$
10: return: $\hat{X} = X^{(k)}$

$\mathcal{P}_{\mathcal{C}}(\cdot)$ computes the least squares projection of its argument onto the set $\mathcal{C}(X)$ defined in (7). Note that we define operations on sequences element-wise, e.g., $\nabla L(X^{(k)}, y(\Theta)) = (\nabla L(X_n^{(k)}, y(\Theta)))_{n=0}^{N-1}$.

The main difficulty with this approach is that not all sequences $\hat{X}$ can be mapped back to valid time-domain signals $\hat{x}$. This is because the analysis in (4) uses overlapping windows, which causes adjacent elements in the sequence $\hat{X}$ to become dependent. This means that individual elements from the sequence $\hat{X}$ cannot be adjusted arbitrarily if we want $\hat{X}$ to have an analog in the time-domain. Therefore, in order to generate valid adversarial examples, we include an additional processing step that projects the sequence $\hat{X}$ onto the space of time-frequency coefficients arising from valid time-domain sequences. This is done using the Griffin and Lim algorithm [25], which seeks to minimise

$$\mathcal{P}_{\mathcal{GL}}(\hat{X}) = \min_{Z \in \mathcal{X}} \sum_{n=0}^{N-1} ||Z_n - \hat{X}_n||_2^2$$

where $\mathcal{X} = \{X = (X_n)_{n=0}^{N-1} : \mathcal{P}_{\mathcal{GL}}(X) = X\}$ denotes the set of all valid sequences. This minimization can be performed using alternating projections, and we have found that in practice it is sufficient to apply a single set of projections. We do this by first rebuilding a complex valued time-frequency representation from the sequence $\hat{X}$

$$U[m, u] =
\begin{cases}
\hat{X}_u / T & [m, u \mod T] \in \Phi_{\mathcal{F}}[m, u], 0 \leq m < D \\
\hat{X}_u / T & [D - m, u \mod T] \in \Phi_{\mathcal{F}}[m, u], D \leq m < L.
\end{cases}$$

where $D = L/2 + 1$ and $\Phi_{\mathcal{F}}[m, u] \triangleq \mathcal{F}(x)$ is the phase from the exemplar’s Fourier transform. The inverse Fourier transform $F^{-1}(U)$ is a time-domain signal, and so the Fourier transform of this signal, $\mathcal{F} \circ F^{-1}(U)$, will yield a valid DFT spectrum that can be used to build a valid input sequence for our (C)DNN, i.e., by replacing $\mathcal{F}(x)$ by $\mathcal{F} \circ F^{-1}(U)$ in (4).

The pseudo-code in Alg. 1 summarises this approach. The algorithm may be terminated when the mean posterior of the target adversarial label exceeds the threshold $R_{\text{min}}$, or after a maximum number of epochs $k_{\text{max}}$ (in which case an adversary cannot be found above the minimum SNR).

Algorithm 2 Train (C)DNN using database of labeled sequences $(X, Y)$ and fast adversarial generation [24], with $\epsilon$ and $\mu$ the gradient descent step sizes for adjusting the adversarial inputs and network weights, respectively.

1: parameters: $\epsilon$, $\mu$
2: init: (C)DNN parameters $\Theta$ to small random weights
3: repeat
4: select $\hat{Y}$ uniformly $\{1, \ldots, K\}^N$
5: $X \leftarrow X + \epsilon \nabla L(X, Y(\hat{\Theta}) \{\text{Generate adversarial ex.}\}$
6: $\Theta \leftarrow \Theta + \mu \nabla L(X, Y(\hat{\Theta}) \{\text{Model update}\}$
7: until Stopping condition

B. Training with adversaries for music audio

As per [54] and [24], we can attempt to use our adversary as a regulariser, and to create systems robust against adversarial inputs. In particular, we create adversaries for the (C)DNN discussed above, and use them to generate a (possibly) infinite supply of new samples during training. The iterative procedure for generating adversaries in Alg. 1 is too slow to be practical for training, which requires on the order of 50 to 200 training epochs. Therefore, we apply the single gradient step procedure suggested in [24]. In our experience, this procedure often generates inputs that confuse the network, although not typically with a high confidence. The pseudo-code in Alg. 2 illustrates our training algorithm, where $(X, Y)$ represent the training data, i.e., the set of input audio sequences and their labels, and $\hat{Y}$ is a set of adversarial labels.

IV. EXPERIMENTAL RESULTS

We can design an adversary (Alg. 1) such that it will attempt to make a system behave in different ways. For instance, an adversary could attempt to perturb an input within some limit (SNR) such that the (C)DNN makes a high-confidence classification ($R_{\text{min}} \approx 1$) that is correct with probability $p$. Another adversary could attempt to make the system label any input using the same label. We can also make an ensemble of adversaries such that they produce adversarial examples that a (C)DNN classifies in every possible way.

We define our adversaries (Alg. 1) using: $R_{\text{min}} = 0.9, \text{SNR} = 15\text{dB}, \mu = 0.1$, and $k_{\text{max}} = 100$, and with the directive to make the (C)DNN correct with probability $p = 0.1$. More concretely, for each test observation, the adversary draws uniformly one of the dataset labels $y_i$, then seeks to find in no more than $k_{\text{max}} = 100$ iterations using step size $\mu = 0.1$ a valid perturbation no larger than $15\text{dB}$ SNR, and which the (C)DNN labels as $y$ with confidence $R_{\text{min}} = 0.9$. Figure 4(a) shows the FoM of the DNN-based classification system in Fig. 4(b), but with input intercepted by this adversary. Note that in this case the classification is performed by the same random forest classifier using the aggregated hidden layer activations, but the adversary is unaware of this. In other words, it is only trying to force the DNN to misclassify inputs that have been subject to minor perturbations. Compared with a normalised
For the same system in Fig. 4(b), and using $R_{\min} = 0.9$, SNR = 15dB, $\mu = 0.1$ and $k_{\max} = 100$, we show in [33] that we able to create adversaries that make the system always right, always wrong, and always select “Jazz.” Table II shows the results of two ensembles of adversaries, each intent on making the system in Fig. 4(b) choose one of every label in GTZAN for the same music with SNR = 15dB, $\mu = 0.1$ and $k_{\max} = 100$. The adversaries of one ensemble insist upon a classification confidence of at least $R_{\min} = 0.5$; and in the other of at least 0.9. These music recordings are the same 30-second excerpts used in [43]. We see that in all case by one, the ensembles are able to elicit high confidence classifications from the system with minor perturbations of the input. We also see that larger perturbations are produced on average when the SNR is reduced, with SNR $\rightarrow 33$ and lessen the SNR constraint to SNR $\rightarrow 15$. The CDNN proved more difficult to fool, but still the adversary is able to significantly reduce the normalised classification accuracy from $R_{\max}$ to $R_{\min}$ in at least one case, the CDNN proved more difficult to fool.

$$\text{Table II}$$

<table>
<thead>
<tr>
<th>Music excerpt</th>
<th>Blues</th>
<th>Classical</th>
<th>Country</th>
<th>Disco</th>
<th>Hip Hop</th>
<th>Jazz</th>
<th>Metal</th>
<th>Pop</th>
<th>Reggae</th>
<th>Rock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Willie Nelson, “A Horse Called Music”</td>
<td>25 (1)</td>
<td>25 (20)</td>
<td>30 (27)</td>
<td>30 (20)</td>
<td>29 (19)</td>
<td>30 (25)</td>
<td>27 (23)</td>
<td>21 (20)</td>
<td>30 (23)</td>
<td>29 (27)</td>
</tr>
<tr>
<td>Simon Mobile Disco, “10000 Horses Can’t Be Wrong”</td>
<td>31 (30)</td>
<td>36 (31)</td>
<td>38 (32)</td>
<td>45 (34)</td>
<td>41 (33)</td>
<td>40 (32)</td>
<td>33 (33)</td>
<td>32 (34)</td>
<td>42 (23)</td>
<td>38 (33)</td>
</tr>
<tr>
<td>Ruben Bandits, “Horse Outside”</td>
<td>27 (27)</td>
<td>27 (27)</td>
<td>36 (29)</td>
<td>42 (31)</td>
<td>38 (29)</td>
<td>34 (28)</td>
<td>32 (28)</td>
<td>37 (29)</td>
<td>36 (29)</td>
<td>35 (29)</td>
</tr>
<tr>
<td>Jethro Tull, “Heavy Horses”</td>
<td>29 (26)</td>
<td>28 (26)</td>
<td>40 (29)</td>
<td>42 (29)</td>
<td>38 (28)</td>
<td>36 (28)</td>
<td>34 (28)</td>
<td>34 (27)</td>
<td>36 (28)</td>
<td>36 (29)</td>
</tr>
<tr>
<td>Count Prince Miller, “Mule Train”</td>
<td>32 (30)</td>
<td>29 (30)</td>
<td>41 (33)</td>
<td>37 (34)</td>
<td>43 (33)</td>
<td>36 (31)</td>
<td>33 (31)</td>
<td>42 (34)</td>
<td>41 (33)</td>
<td>33 (33)</td>
</tr>
<tr>
<td>Rolling Stones, “Wild Horses”</td>
<td>30 (22)</td>
<td>32 (24)</td>
<td>37 (25)</td>
<td>40 (25)</td>
<td>31 (22)</td>
<td>34 (25)</td>
<td>31 (26)</td>
<td>32 (23)</td>
<td>37 (25)</td>
<td>37 (26)</td>
</tr>
</tbody>
</table>

Fig. 4. For the DBN-based classifier in Fig. 2(b) and the CDNN in Fig. 3(d), with all input inserted by an adversary intent on making the maximum posterior posterior prior with probability $p = 0.1$. This adversary, $R_{\min} = 0.5$, SNR = 15dB, $\mu = 0.1$, and $k_{\max} = 100$. Sub-captions show the resulting SNR (mean ± standard deviation) for the adversarial test sets of GTZAN ($N = 290$) and LMD ($N = 640$).
We now perform an experiment to compare (C)DNNs trained with adversarial examples (as per Alg. 2) to the systems in Fig. 3(b,d). To do this we test the response of these systems against an adversary aimed at always eliciting an incorrect response. (This is different from the adversary used above, which seeks to make the system correct with probability $p = 0.1$.) For this experiment, we set $R_{\min} = 0.5$ and SNR $= -300$ dB in order to allow arbitrarily large perturbations to force misclassifications. Table III illustrates the results of this experiment from which we observe several interesting results. Column 1 shows the normalized accuracy on the original test set (with no adversary present). We see that training against adversarial examples leads to a slight deflation in accuracy on new test data. Column 2 shows the normalized accuracy of these systems against our adversary intent on forcing a 100% error rate. We see that the CDNN systems are more robust to this adversary, and that the systems trained against adversarial examples confer little to no advantage. Column 3 shows the average perturbation size of the adversarial examples that led to misclassifications. We notice that larger perturbations (corresponding to lower SNRs) were required to get the CDNN systems to misclassify test inputs. The minimum SNR produced was 0.11 dB, while the maximum was 47.6 dB.

The results of this experiment point to the conclusions that a) the CDNN systems are more robust to this adversary; and b) training against adversarial examples (contrary to what we hypothesized) does not seem reduce the misclassification rate against new adversarial examples. A possible explanation for the latter results is that, due to the high-dimensional nature of the input space, the set of possible adversarial examples is densely packed, so that training on a small number of these points is not sufficient to allow the systems to generalize to new adversarial examples.

V. Discussion

Returning to the broadest question motivating our work, we seek to measure the contribution of deep learning to music content analysis. The previous sections describe a series of experiments we have conducted using deep learning systems of a variety of architectures, which we have trained and tested in two different partitions of two benchmark music datasets. We have evaluated the robustness of these systems to an adversary that has complete knowledge of the classifiers, and have also investigated the use of an adversary in the training of deep learning systems.

Our experimental results in Fig. 2 and Table I are essentially reproductions of those reported in [44]. Based on the results of their experiments with random partitionings of GTZAN, Sigia et al. [44] claim that their DNN-based systems learn features that “better represent the audio” than standard or “hand-crafted” features, e.g., those referenced in [30] like MFCCs. Similar conclusions are made about the deep learning systems in [28], also based on experiments using a random partitioning of GTZAN. However, we see in Fig. 2 and Table I that when we consider the faults in the GTZAN dataset and partition it along artist lines, as for the LMD dataset in Fig. 3 our deep learning systems perform significantly worse. This is
This motivates the question of whether DNN-based systems really do perform better than that of a classifier using standard, low-level and “hand-crafted” features. To examine this, we build baseline systems that use low-level features, and train and test them in the same fault-filtered partition of GTZAN as in Fig. 2(b), and the artist-filtered partition of LMD as in Fig. 3(b,d). Mimicking [28], [44], we compute these features based on a short-time analysis using 46ms frames hopped by 50%. From each frame we extract the first 13 Mel-frequency cepstral coefficients (MFCCs) and zero-crossings, and compute their mean and variance over five-second texture windows (which are also hopped by 50%). We combine the features of the training and validation sets of the fault-filtered partition of GTZAN, and the artist filtered partition of LMD.

Both systems use a minimum Mahalanobis distance classifier, and assign a class by majority vote from the classifications of the individual texture windows. Figure 6 shows the FoM produced by these baseline systems. We see that for GTZAN it actually reproduces more ground truth than the DNN in Fig. 2(b) and all but one in Table I. Our simple baseline system for LMD reproduces much less ground truth than the (C)DNN in Fig. 3(b,d). Nonetheless, we have no reason to accept the conclusion that deep learning features “perform better” than “hand-crafted” features for the particular architectures considered here and those in [28], [44]. Different experiments are needed to address such a conclusion.

A tempting conclusion is that since the normalised classification accuracies in Figs. 2(b) and 3(d) are extremely unlikely to arise by chance (p < 10^{-62} for GTZAN and p < 10^{-290} for LMD by a Binomial test) it is therefore entirely reasonable to reject the hypothesis that our (C)DNN are choosing outputs at random. Hence, one might argue that these (C)DNN must have learned features that are “relevant” to music genre recognition [28], [31], [44]. This argument appears throughout the MIR research discipline [49], and turns on the strong assumption that there are only two ways a system can reproduce the ground truth of a dataset: by chance or by learning to solve a specific problem thought to be well-posed by a cleanly labeled dataset [51]. In fact, there is a third way a system can reproduce the ground truth of a music dataset: by learning to exploit characteristics shared between the training and testing datasets that arise not from a relationship in the real world, but from the curation and partitioning of a dataset in the experimental design of an evaluation [48], [49], [53]. Since the evaluations producing Figs. 2 and 3, as well as all results in [28], [44], not to mention a significant number of published studies in MIR [49], do not control for this third way, we cannot validly conclude upon the “relevance” of whatever has been learned by these music content analysis systems.

A notion of this problem is given by the significant decreases in the FoM we measure when partitioning GTZAN and LMD along artist lines. By doing so, we are controlling for some independent variables that a system might be exploiting to reproduce ground truth, but which arguably have little relevance to the high-level labels of the dataset [49]. More concretely, consider that all 100 excerpts labeled Pop in GTZAN come from recordings of music by four artists, 25 from each artist. If we train and test a system on a random partition of GTZAN, we cannot know whether the system is recognising Pop, recognising the artist, or recognising other aspects that may or may not be related to Pop. If we train a system instead with Pop excerpts by three artists, test with the Pop excerpts by the fourth artist, then we might be testing something closer to Pop recognition. This all depends on defining what knowledge is relevant to the problem.

A common retort to these arguments is that a system should be able to reproduce ground truth “by any means.” One thereby defines “relevant knowledge” as any correlations that helps a
system reproduce an amount of dataset ground truth that is inconsistent with chance. However, this can lead to circular reasoning: system X has learned “relevant knowledge” because it reproduces Y amount of ground truth; system X reproduces Y amount of ground truth because it has learned “relevant knowledge.” It is also adrift to one of the major aims of research in music content analysis [14]: “to make music, or information about music, easier to find.” If a music content analysis system is describing music in ways that do not align with those of its users, then its usability is in jeopardy no matter its FoM in benchmark datasets [42], [56]. Finally, this means that the problem thought to be well-posed by a cleanly labeled dataset can be many things simultaneously — which leads to the problem of how to validly compare apples and oranges [51]. In other words, why compare systems when they are solving different problems? This also applies to the comparisons above with the FoM in Fig. 6.

While we have no idea whether our (C)DNN systems in Fig. [3] are exploiting “irrelevant” characteristics in LMD, our experimental results with adversaries in Figs. 4 and 5 and Tables 1 and 11 indicate that their decision machinery is incredibly sensitive in very strange ways. Our adversaries are able to fool the high-performing deep learning systems by perturbing their input in minor ways. Auditioning the results in Table 1 show that while the music in each recording remains exactly the same, and the perturbations are very small, the DNN is nearly always fooled into choosing with high confidence every class it has supposedly learned. The CDNN is similarly defeated by our adversary; however, it is quite notable that it requires perturbations of far lower SNR than does the DNN. We are currently studying the reasons for this.

Our application of adversaries here is close to the “method of irrelevant transformations” that we apply in [48], [52], [53] to assess the internal models of music content analysis systems, and to test the hypothesis, “the system is using relevant criteria to make its decisions.” In [48], we take a brute force approach whereby we apply random but linear time-invariant and minor filtering to inputs of systems trained in three different music recording datasets until their FoM becomes perfect or random. We also make each system apply every one of its classes to the same music recordings in Table 11. In [53], we instead apply subtle pitch-preserving time-stretching of music recordings to fool a deep learning system trained in the benchmark music dataset BALLROOM [20]. We find that through such a transformation we can make the system perform perfectly or no better than random by applying tempo changes of at most 6% to test dataset recordings. We find a similar result for the same kind of deep learning system but trained in LMD [52].

Our adversary in Alg. 1 moves instead right to the achilles heel of a deep learning system, coaxing it to behave in arbitrary ways for an input simply by making minor perturbations to the sampled audio waveform that have no effect on the music content it possesses. We observe in Fig. 5 and auditioning Table 11 that the low- to mid-frequency content of adversarial examples differs very little from the original recordings, but find more significant differences in the high-frequency spectra. This suggests that the distribution of energy in the high-frequency spectrum has significant impact on the decision machinery of our (C)DNN. The apparent high relevance of such slight characteristics in proportion to that of the actual musical content of a music recording does not bode well for one of the most important aims of machine learning: generalisation.

As observed by Goodfellow et al. [24] in their deep learning systems taught to recognise objects in images, the impressive FoM we measure of our deep learning systems may be merely a colourful “Potemkin village.” Employing an adversary to scratch a little below the surface reveals the FoM to be curiously hollow. A system that appears to be solving a complex problem but actually is not is what we term a “horse” [48], which is a nod to the famous horse Clever Hans: a real horse that appeared to be a capable mathematician but was merely responding to involuntary cues that went undetected because his public demonstrations had no validity to attest to such an ability. Measuring the number of correct answers Hans gives in an uncontrolled environment does not give reason to conclude he comprehends what he appears to be doing. It is the same with the experiments we perform above with systems labelling observations in GTZAN and LMD. In fact, Goodfellow et al. [24] come to the same conclusion: “The existence of adversarial examples suggests that ... being able to correctly label the test data does not imply that our models truly understand the tasks we have asked them to perform.” [24]. This observation is now well-known in MIR [47]–[50], but deserves to be repeated.

VI. Conclusion

In this article, we have shown how to adapt the adversary of Szegedy et al. [54] to work within the context of music content analysis using deep learning. We have shown how our adversary is effective at fooling deep learning systems of different architectures, trained on different benchmark datasets. We find our convolutional networks are more robust against this adversary than our deep neural networks. We have also sought to employ the adversary as part of the training of these systems, but find it results in systems that remain as sensitive to the same adversary.

It is of course not very popular for one to be an “adversary” to research, moving quickly to refute conclusions and break systems reported in the literature; however, we insist that breaking systems leads ultimately to progress. Considerable insight can be gained by looking behind the veil of performance metrics in an attempt to determine the mechanisms by which a system operates, and whether the evaluation is any valid reflection of the qualities we wish to measure. Such probing is necessary if we are truly interested in ascertaining what a system has learned to do, what its vulnerabilities might be, how it compares to competing systems supposedly solving the same problem, and how well we can expect it to perform when used in real-world applications.
REFERENCES


