DEEP SYNTHESIZER PARAMETER ESTIMATION

Oren Barkan and David Tsiris

Tel Aviv University

ABSTRACT

Manual tuning of synthesizer parameters to match a specific sound can be an exhaustive task. This paper proposes an automatic method for synthesizer parameters tuning to match a given input sound. The method is based on strided Convolutional Neural Networks and is capable of inferring the synthesizer parameters configuration from the input spectrogram and even from the raw audio. The effectiveness of our method is demonstrated on a subtractive synthesizer with frequency modulation. We present experimental results that showcase the superiority of our model over several baselines. We further show that the network depth is an important factor that contributes to the prediction accuracy.

Index terms – deep *parameter estimation, deep sound synthesis.*

1. INTRODUCTION AND RELATED WORK

The art of sound synthesis is a challenging task that mainly reserved to sound designers and engineers [1]. Nowadays, synthesizers play a key role in electronic music production. Typically, music producers are equipped with a set of sound banks per synthesizer. These sound banks are essentially a collection of different configurations (patches) of the synthesizer parameters. Each configuration was carefully tuned by an expert and produces a different type of sound.

During the last decade, the emergence of deep learning advanced the state of the art in various fields [16]-[24]. Specifically, convolutional neural networks (CNN) were found to be extraordinary learners for music related tasks [2]-[6]. In this paper, we investigate the problem of estimating the synthesizer parameter configuration that best reconstructs a source audio signal. We assume the source audio signal is generated from the same synthesizer (intra domain) by a hidden parameters configuration and leave the cross domain problem for future investigation. Therefore, our goal is to reveal the hidden parameters configuration that was used to generate the source signal. This is particularly useful in scenarios where music artists are interested in replicating sounds that were generated by other artists / sound designers using the same synthesizer (assuming no patch is released). To this end, we propose to train a 2D strided CNN to predict the synthesizer parameter configuration from the Short Time Fourier Transform (STFT) spectrogram [1] of the input audio signal. Moreover, we show that a CNN is capable of performing end to end learning, directly from the raw audio to the synthesizer parameters domain. This is done by adding several convolutional layers that are designed to learn an **alternative** representation for the STFT.

The proposed pipelines are depicted in Fig. 1. In Fig.1 (a), an input audio signal is transformed to a STFT spectrogram matrix, which is then fed to a CNN. The CNN analyzes the spectrogram and predicts a parameter configuration. Finally, the synthesizer is configured according to the predicted parameters values and synthesizes the output audio signal. In Fig. 1(b), a CNN performs end to end learning and predicts the parameter configuration directly from the raw audio. In addition, we compare the performance of these models against two other types of fully connected (FC) neural network models: the first type is a FC network that receives a Bag of Words (BoW) representation of the spectrogram as input. The second type is a FC network that receives a set of complex handcrafted features [10] that capture spectral and temporal properties.

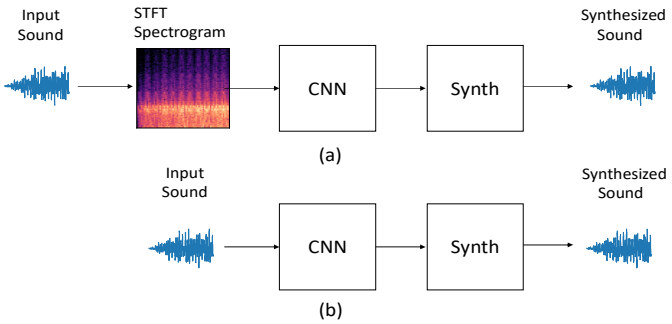

Figure 1*. The proposed pipelines. (a) The STFT spectrogram of the input signal is fed into a CNN that predicts the synthesizer parameter configuration. This configuration is then used to produce a sound that is similar to the input sound. (b) End to end learning. A CNN predicts the synthesizer parameter configuration directly from the raw audio.*

The audio signals that are used for training and testing the models are generated by synthesizer parameter configurations that are randomly sampled, i.e. for each synthesizer parameter, we define an effective range of valid values and sample the parameter value from this range, uniformly. Hence, each configuration in the dataset is obtained by a set of (parameter, sampled value) pairs.

We present a comprehensive investigation of various network architectures and demonstrate the effectiveness of the proposed method in a series of quantitative and qualitative experiments. Our findings show that 1) the network depth is an important factor which contributes to the prediction accuracy and 2) a spectrogram based CNN of a sufficient depth outperforms its end to end counterpart, while both CNN variants significantly outperform FC networks.

Several attempts has been made to apply traditional machine learning techniques to physical modeling of musical instruments such as bowed and plucked strings [7]–[9]. However, we focus on subtractive synthesis with frequency modulation (FM) [1] that is more common in electronic music production and enables the creation of extremely diversified sound banks. This is in contrast to physical modeling synthesis that is mainly used for creating real-sounding instruments, as it is programmed to make characteristic distinctions between various aspects of the instrument being created.

Itoyama and Okuno [10] proposed to apply a multiple linear regression from a set of handcrafted features to the synthesizer parameter domain. Our approach, differs from [10] by two main aspects: first, we solve a classification task rather than a regression. Second, we propose deep CNN architectures with non-linear gates. These networks have a sufficient capacity to learn complex filters (weights) that are capable of capturing important patterns and yet generalize well. This eliminates the need for handcrafted features and enables the use of STFT spectrogram / raw audio as input without further manipulations. We validate this claim, empirically, by comparing the CNN models to linear and non-linear FC models that use the handcrafted features [10].

2. SYNTHESIZER ARCHITECTURE AND PARAMETERS

This section describes the synthesizer architecture that is used in this work. The synthesizer is implemented using JSyn [11], an open source library that provides audio synthesis API for Java. The reason we use JSyn is twofold: most of commercial synthesizers usually do not provide an API for generating sounds, programmatically. This makes the dataset generation process impractical (our dataset contains ~200K samples). Second, commercial synthesizers are not open source and quite expensive. Therefore, we avoid of using these synthesizers in order to make our research reproducible.

In similar to the majority of modern synthesizers, we employ subtractive and FM synthesis [1]. The synthesizer architecture is a cascade of four components. The first component consists of four oscillators, each produces a different waveform type [1]: sine, saw, square and triangle. All oscillators are frequency modulated by a sinusoidal waveform. An oscillator function is defined as $y_w(f, v, A, B) = Ax_w\big(2\pi ft + B\sin(2\pi vt)\big)$ where f, v, A, B are the carrier frequency, modulation frequency, carrier amplitude and modulation amplitude, respectively. $w \in W$ is the waveform type with $W = \{sin, saw, tri, sqr\}$ that correspond to the abovementioned waveforms. Note that each oscillator y_w is associated with its **own** set of f_w, v_w, A_w, B_w parameters. Finally, the outputs from all oscillators are summed to $y_{osc} = \sum_{w \in W} y_w$. Therefore, the number of parameters in the y_{osc} component is 16.

The second component is the *Attack Decay Sustain Release* (ADSR) envelope generator [1] $y_{env}(x, a, d, s, r)$. This component controls the amplitude of the input x at any point in the signal duration. The contour of the ADSR envelope is specified using four parameters: a (*Attack*) is the time taken for initial run-up of level from zero to peak, beginning when the key is first pressed. d (*Decay*) is the time taken for the subsequent run down from the attack level to the designated sustain level. s (*Sustain*) is the level during the main sequence of sound's duration, until the key is released. r (*Release*) is the time taken for the level to decay from the sustain level to zero after the key is released.

The third component is the filter $y_{lp}(x, f_{cut}, q)$ that consists of a low-pass filter together with a resonance [1]. Setting a cutoff frequency f_{cut} ensures that all frequencies above f_{cut} are cut. The resonance parameter q determines a narrow band of frequencies near f_{cut} that are amplified. The last component in the chain is the gater - a Low Frequency Oscillator (LFO) that performs amplitude modulation to the input using a square wave with a frequency f_{gate}: $y_{gate}(x, f) = ((1 + x_{sqr}(2\pi f_{gate} t))/2)x(t)$. Thus, the synthesizer function has 23 parameters and is given by $y_{gate} * y_{lp} * y_{env} * y_{osc}$, where $*$ stands for the function composition.

3. DATASET GENERATION

For each synthesizer parameter (described in Section 2), we perform a quantization to a set of 16 levels and treat each level as a different class. Hence, we formulate the parameter estimation task as a classification problem rather than a regression - our model aims at predicting for each parameter the correct class (value). This enables the use of the binary cross entropy loss [6] that is easier to optimize than the L2 loss (a similar observation is made in [12]). Moreover, the classification formulation naturally allows for a specific measure (Section 5) that enables better quantification and understanding of the parameter estimation accuracy.

The range of the carrier frequency f is quantized according to $f = 2^{n/12} \times 440Hz$ with $n \in \{0..15\}$. This produces frequencies that correspond to the 16 consecutive musical notes $A_4 - C_6$. The rest of the synthesizer parameters ranges are quantized evenly to 16 classes according to the following ranges: the amplitudes and ADSR envelope parameters A_w, a, d, s, r are in [0.001, 1], the modulation amplitudes B_w are in [0, 1500], the modulation frequency, gating frequency, cutoff frequency and resonance $v_w, f_{gate}, f_{cut}, q$ are in [1, 30], [0.5, 30], [200, 4000], [0.01, 10], respectively. The described sampling pattern avoids sounds generation that are indistinguishable by human hearing and better matches a linear perception.

The dataset consists of (g, h) pairs, where h is the label vector that corresponds to a specific synthesizer parameter configuration and g is the raw audio signal produced by the configuration that is associated with h. The generation process of a single pair (g, h) works as follows: for each synthesizer parameter j, we sample a class from a uniform categorical random variable $u \in \{0, \dots, 15\}$ and create an one-hot encoding vector $h_i \in \{0,1\}^{16}$ for the sampled class. Then, all vectors are concatenated to a supervector $h = [h_1, \dots, h_{23}] \in \{0,1\}^{368}$ that contains the one-hot encoding for each parameter in the corresponding section in h. Finally, the synthesizer parameters are configured according to values that correspond to the sampled classes in h and an output audio signal in a duration of 1 second with a sampling rate of 16384Hz $g \in [-1,1]^{16384}$ is produced.

Recall that we consider two types of CNN architectures: an end to end (Fig. 1(b)) and a spectrogram based (Fig. 1(a)), where the latter requires further transformation over g. Therefore, the STFT spectrogram of g is computed with half overlapping windows of size 512 to produce a matrix $S \in \mathbb{R}^{257 \times 64}$. This matrix contains 64 vectors in size of 257 that correspond to the absolute of the Fourier Transforms (FT) of 64 consecutive time windows. Note that the application of FT produces 512 complex values. However, we discard ~half of them since the FT of real signals is conjugate symmetric. This process is repeated 200k times to produce two datasets: $D_{E2E} = \{(g_k, h_k)\}$ and $D_{STFT} = \{(S_k, h_k)\}$ that are used for training the end to end and STFT based CNNs, respectively.

4. MODELS AND METHODOLOGY

In this section, we describe the CNN components from the two different pipelines that appear in Fig. 1. The goal of these components is to learn a function from the STFT / raw audio domains to the synthesizer parameter domain. To this end, we propose to train CNNs using the datasets described in Section 3 to predict the correct parameter classes (values) from the STFT matrix / raw audio. As a baseline, we further consider the replacement of the CNNs with Fully Connected (FC) neural networks. All models have output layers in size of 368 with sigmoid activations (to match h's dimension). Note that an alternative is to apply 23 separated softmax activations that are fed into 23 categorical cross entropy loss functions (for each synthesizer parameter, separately) and compute the final loss as the summation of the 23 loss functions. While this alternative approach seems more natural, in our initial experiments, it performs worse.

In order to isolate the effect of network depth, we restrict all networks to have the **same** number of trainable parameters, regardless of their depth. This ensures that better results, when obtained using a deeper model, are indeed due to the increase in depth. Yet, we did conduct an initial investigation to find a saturation point in which a further increase in number of trainable parameters (model capacity) materializes to a marginal contribution to the model accuracy. In what follows, we describe all of the evaluated models.

4.1. Fully Connected Neural Networks

A FC neural network is a feed forward network in which each neuron is connected to all neurons in the previous layer [6]. This type of networks expect to have 1D input (vector). Since the training instances are STFT matrices / raw audio signals, the trivial choice is to feed them to the FC network as is (flattened). However, in our initial experiments, both approaches produced poor results. This is since both flattened STFT matrices and raw audio signals are not time invariant and of an extremely high dimension (~16K).

In order to alleviate this problem, we propose to use two different types of input representation to the FC models: the first type is based on the STFT matrix. Specifically, we first apply PCA to the STFT frequency axis to produce a STFT-PCA matrix with a reduced frequency-PCA dimension of 64 (while retaining 97% of the variance). Then, we learn a Bag of Words (BoW) [13] representation of the STFT-PCA matrices. To this end, we Vector Quantize [13] the STFT matrices using K-means [13] with $K = 1000$ and assign each row vector in the STFT-PCA matrix to its closest centroid. This produces a count vector in size 1000, in which the i-th entry counts the number of vectors that were assigned to i-th centroid. Finally, we convert the counts to probabilities by sum normalization. The result is a time invariant representation of a reduced dimensionality.

We consider four different FC model architectures that use the BoW representation as input: the first model has a single hidden layer with linear activations and is dubbed FC Linear (note that in this work, we define the number of layers in a network, as the number of hidden layers in between the input and output layers). Hence, FC Linear is equivalent to a logistic regression model with a hidden layer. The three other models are dubbed FC1, FC2 and FC3 and have 1, 2 and 3 hidden layers with ReLU [6] activations, respectively. We observed that further increase in network depth to more than 3 layers did not contribute to improved performance.

The second type of input representation that we consider is a set of complex handcrafted features [10] that are computed from the raw audio signal. These features capture both spectral properties and temporal variations in the signal and form a 319,200 dimensional supervector. This supervector is then reduced to dimension of 1000 by the application of PCA. The reader is referred to Section 2 in [10] for a detailed description of the feature extraction process.

We consider two different FC model architectures that use the input representation from [10]. The first is a linear FC model that is equivalent to the one from [10], but uses a classification output layer instead of a regression (In our initial experiments, we tested the regression models from [10] and observed they underperform classification models). We dub this model 'HC' (handcrafted). In addition, we investigated whether a nonlinear FC model can benefit from using the features from [10]. We found that a FC network with 3 ReLU activated layers and a dropout in between obtained the largest improvement over HC [10], while further increase in depth or number of trainable parameters results in overfitting with worse values of validation loss. We dub this model 'HC3'. Note that HC3 is an improvement we suggest, to showcase the potential gain that is obtained by using the features from [10] with deep neural networks. For all models in this section, a dropout [6] is applied after each hidden layer in order to avoid a severe overfitting. All FC models have 1.2M trainable parameters and are fully detailed in [15].

4.2. Strided Convolutional Neural Networks

The second type of models we use are 2D CNNs as these were found to improve on the state of the art in music and audio related tasks [5]. Different from [5], we do not perform any type of pooling operations. Instead, we use strided convolutions layers [6], as we found this approach to significantly outperform the traditional setup of ordinary convolutions layers with pooling in between.

Two types of CNNs are considered: the first type is a spectrogram based 2D CNN that receives the STFT matrix as input. This network learns filters that analyze the input in both frequency and time axes, simultaneously. We investigate seven different spectrogram based CNNs. The first six models have the same number of 1.2M trainable parameters, but vary by network depth. These models are dubbed Conv1, …, Conv6 and have 1,…,6 2D strided convolutional layers, respectively. The seventh CNN is dubbed Conv6XL and has 6 strided convolutional layers, but 2.3M trainable parameters. The reason we further include Conv6XL in the evaluation is to check whether increasing the model capacity, in terms of number of trainable parameters, further contributes to the prediction accuracy.

The second type of CNN is an end to end CNN that receives the raw audio signal as input. This network is dubbed ConvE2E and further aims at learning a set of filters that produce a transformation on the raw audio, which serves as an **alternative** to the STFT filters. To this end, the first four layers in the ConvE2E model are designed to transform the 16K dimensional input signal to a matrix in the exact same size of the STFT matrix (64x257). These four layers are 1D strided convolutional layers that operates on the time axis **only**. This is followed by additional six 2D strided convolutional layers that are identical to those of the Conv6 model. Due to the extra four layers, ConvE2E has 1.9M trainable parameters. Finally, all CNNs have an additional FC hidden layer in between the last convolutional and the output layers. The exact CNN architectures are fully detailed in [15].

5. EXPERIMENTAL SETUP AND RESULTS

In this section, we describe the experimental setup, evaluation measures and present quantitative and qualitative results. The reported results are obtained using 10 fold cross validation on the datasets that are described in Section 4. All models are optimized w.r.t. the binary cross entropy loss [6] using ADAM [14] optimizer with mini batch size of 16 for 100 epochs.

The training and validation loss per model are displayed in Fig. 2. We see that most of the CNN models obtain significantly lower validation loss values than their FC and HC counterparts. Specifically, Conv6XL and Conv6 perform the best and on par with each other. Therefore, we conclude that increasing the number of trainable parameters from 1.2M to 2.3M has a negligible effect for CNNs of a sufficient depth. The largest decrease in loss is obtained when moving from Conv1 to Conv2 and then from Conv2 to Conv3. We believe that this is since Conv1 and Conv2 uses strides values that results in too aggressive subsampling. ConvE2E significantly underperforms Conv4, Conv5, Conv6 and Conv6XL, but is on par with Conv3 and outperforms all FC and HC models. Hence, we conclude that the STFT spectrogram contains further crucial information that ConvE2E fails to extract from the raw audio. Yet, ConvE2E manages to learn filters that are better than the complex handcrafted features from [10], even when combined with a deep FC network (HC3). Among the FC models, FC2 and FC Linear perform the best and the worst, respectively. Note that we omit the loss graphs of FC1 and FC3 as these almost completely overlap with FC2. Examining the HC models, we observe the following trends: HC which is a linear model outperform all non-linear FC models and is on par with Conv1. This can be explained by the fact that HC uses handcrafted features that are far more complex than the STFT spectrogram and its BoW representation. HC3 significantly outperforms HC and Conv1, but underperforms the rest of the Conv models including ConvE2E.

Figure 2 further shows that network depth plays an important factor: the deeper the network the better it manages to learn. Specifically, we observe that a large reduction in loss is obtained by moving from HC to HC3, despite the fact both models have the same number of trainable parameters. This is another evidence for the importance of network depth, which results in additional nonlinear transformations. However, the contribution by depth becomes marginal, when using more than 5, 2 and 3 layers for CNN, FC and HC models, respectively. In terms of generalization, FC1, FC2, FC3 and HC3 start overfitting after ~20 epochs. We tried to alleviate this problem by increasing the dropout values and applying L2 regularization, but this resulted in higher values of validation loss. In the case of CNN models, no overfitting is observed, despite the fact that no regularization is applied. This can be explained by the nature of CNNs that enables weight sharing and hence less tend to overfit.

Table 1. Reconstruction quality for the evaluated models in frequency domains. ρ values are x100.

Measure/Model	Conv6XL	Conv6	Conv5	Conv4	ConvE2E	HC3	HC [10]	FC2	FC Linear
ρ mean (STFT)	92.04	91.38	91.09	90.97	88.33	88.04	84.62	77.22	75.24
ρ median (STFT)	95.61	95.39	95.1	94.99	93.41	92.59	88.53	82.32	80.83
ρ mean (FT)	76.13	74.28	73.46	73.29	71.36	64.39	59.68	58.18	54.74
ρ median (FT)	90.22	88	87.4	86.29	80.61	68.07	59.73	57.19	51.15

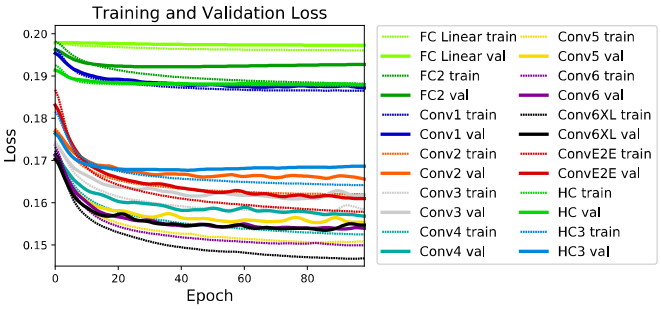

***Figure 2**. Training and validation loss graphs for all models.*

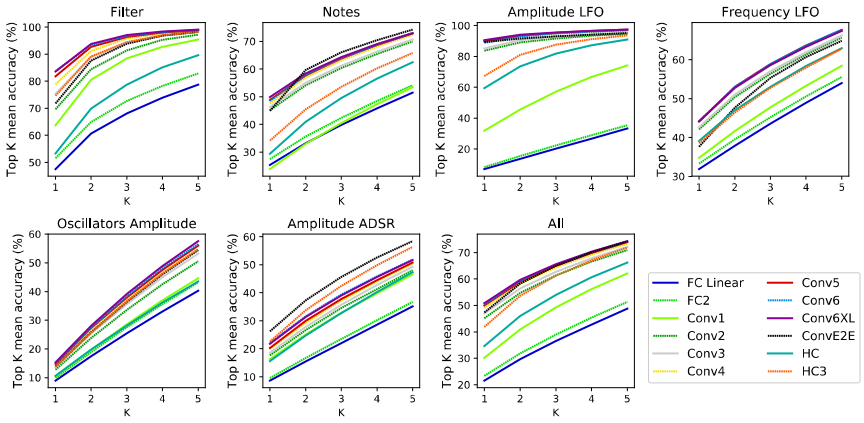

***Figure 3**. Top-k mean accuracy graphs per parameters group.*

Our goal is to measure the reconstruction quality in terms of human hearing. As we are not aware of any quantitative measure that correlates well with human hearing, we propose several additional measures to evaluate the models performance in multiple resolutions: accuracy per group of parameters and reconstruction quality in the frequency domain.

The second measure we use is the top-k mean accuracy. For a given test example, the top-k accuracy function outputs 1 if the correct class rank is among the top k predicted classes by the model and 0 otherwise. The top-k mean accuracy is obtained by computing the top-k accuracy for each test example and then taking the mean across all examples. We compute the top-k mean accuracy per synthesizer parameter for $k = 1, \dots, 5$. Then, we group the synthesizer parameters according to functionality groups: Filter group that contains the parameter f_{cut}, q . Notes, Amplitude LFO, Frequency LFO and Oscillators Amplitude groups that contain the parameters f_w, B_w, v_w, A_w for all w, respectively. The last two groups are the Amplitude ADSR group that contains the parameters a, d, s, r and the All group that contains all the synthesizer parameters. For each combination of model and parameters group, we compute the mean of the top-k mean accuracy across all parameters in that group and for all k values. The reason we divide the parameters into different groups is in order to inspect the models performance w.r.t. each functional in the synthesizer. Figure 3 presents top-k mean accuracy as a function of k for each combination of parameter group and model. The best accuracy values are obtained for the Filter and Amplitude LFO groups, where the margin between the CNN and FC models is the largest. Furthermore, in all graphs the CNNs significantly outperform the FC models. Finally, we observe that for the Amplitude ADSR parameter group, ConvE2E produces the best accuracy graph followed by HC3 as the second best. This is directly related to the ability of these models to predict the attack parameter significantly better than the other models. An interesting pattern is further observed in the Notes graph: while ConvE2E underperforms Conv4 – Conv6XL for k = 1. It becomes the champion for k > 1. This means that ConvE2E manages to infer a better ranking of musical notes when considering the predictions beyond the top-1.

Next, we investigate how well the different models reconstruct the input signals in the frequency domain. The evaluation focuses on a subset of the models that performed the best among each model type (Linear FC, non-linear FCs, HCs and CNNs). Specifically, we compute for each signal x_i in the test set its STFT spectrogram S_i. Then, S_i is fed to the model to produce the synthesizer parameter configuration, which is then used to synthesize an output signal x_o using the synthesizer. Finally, we compute the STFT spectrogram S_o using x_o. Note that in the case of ConvE2E and HC models, x_i is used as input (HC models further require the preprocessing of [10] on x_i).

The spectrogram reconstruction quality is measured by the Pearson Correlation Coefficient (PCC) $\rho(x,y) = cov(x,y)/\sigma_x\sigma_y$. Since ρ is defined over vectors rather than matrices, we flatten S_i and S_o to vectors by performing rows concatenation and then compute their PCC value. In addition, we evaluate the reconstruction quality in the Fourier domain. To this end, we compute the PCC over the absolute of the Fourier Transforms (FT) of x_i and x_o. Different from the STFT, which provides the frequency information per short time frames, the FT provides a global representation that aggregates the frequency information from the entire signal.

Table 1 presents the mean and median values of ρ (x100) that were computed over the test set for each combination of domain and model. We observe that the same trends from Figs. 2 and 3 also exist in Tab. 1. Spectrogram based CNNs performs the best ordered by their depth. ConvE2E outperforms the HC3 and HC models. FC models perform the worst. Hence, we conclude that network depth contributes to the reconstruction quality as well.

Lastly, audio and visual examples of reconstructions (for samples that are drawn randomly from the test set) are provided in [15]. These examples correlates well with the trends that exist in Tab. 1 and further demonstrate the effectiveness of the proposed models.

6. CONCLUSION

We proposed end to end and spectrogram based CNNs for the synthesizer parameter estimation task. Empirical results show that the proposed models outperform HC and FC models. Network depth is found to be an important factor that contributes to the prediction accuracy. In the future, we plan to extend the evaluation to the cross-domain scenario and investigate very deep models such as [25, 26]. We further plan to include the synthesizer function in the backpropagation process, in order to train models to reconstruct the STFT directly.

7. REFERENCES

[1] M. Russ, *Sound Synthesis and Sampling*. 2009.

[2] E. J. Humphrey, J. P. Bello, and Y. LeCun, "Moving Beyond Feature Design: Deep Architectures and Automatic Feature Learning in Music Informatics," *International Society for Music Information Retrieval Conference (ISMIR)*, no. Ismir, pp. 403–408, 2012.

[3] E. J. Humphrey, J. P. Bello, and Y. Lecun, "Feature learning and deep architectures: New directions for music informatics," *Journal of Intelligent Information Systems*, vol. 41, no. 3, pp. 461–481, 2013.

[4] S. Dieleman and B. Schrauwen, "End-to-end learning for music audio," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2014, pp. 6964–6968.

[5] K. Choi, G. Fazekas, and M. Sandler, "Automatic tagging using deep convolutional neural networks," *International Society for Music Information Retrieval Conference*, pp. 805–811, 2016.

[6] A. Goodfellow, Ian, Bengio, Yoshua, Courville, "Deep Learning," *MIT Press*, 2016.

[7] J. Riionheimo and V. Välimäki, "Parameter estimation of a plucked string synthesis model using a genetic algorithm with perceptual fitness calculation," *Eurasip Journal on Applied Signal Processing*, vol. 2003, no. 8, pp. 791–805, 2003.

[8] A. W. Y. Su and S. F. Liang, "A class of physical modeling recurrent networks for analysis/synthesis of plucked string instruments," *IEEE Transactions on Neural Networks*, vol. 13, no. 5, pp. 1137–1148, 2002.

[9] M. Sterling and M. Bocko, "Empirical physical modeling for bowed string instruments," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2010, pp. 433–436.

[10] K. Itoyama and H. G. Okuno, "Parameter Estimation of Virtual Musical Instrument Synthesizers," in *Proceedings of ICMC*, 2014, pp. 95–101.

[11] http://www.softsynth.com/jsyn

[12] Van Den Oord Aaron and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," *arXiv preprint arXiv:160903499*, 2016.

[13] C. M. Bishop, *Pattern Recognition and Machine Learning*, vol. 4, no. 4. 2006.

[14] D. P. Kingma and J. L. Ba, "Adam: a Method for Stochastic Optimization," *International Conference on Learning Representations 2015*, pp. 1–15, 2015.

[15] https://github.com/deepsynth/deepsynth

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

[17] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," in *{IEEE} International Conference on Acoustics, Speech and Signal Processing, {ICASSP} 2013, Vancouver, BC, Canada, May 26-31, 2013*, 2013, pp. 6645–6649.

[18] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A Neural Probabilistic Language Model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, Mar. 2003.

[19] W. Yih, K. Toutanova, J. C. Platt, and C. Meek, "Learning Discriminative Projections for Text Similarity Measures," in *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, 2011, pp. 247–256.

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in *Advances in Neural Information Processing Systems 26*, 2013

[21] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural Language Processing (Almost) from Scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, Nov. 2011.

[22] Y. Kim, "Convolutional Neural Networks for Sentence Classification," *Proc. 2014 Conf. Empir. Methods Nat. Lang. Process. (EMNLP 2014)*, pp. 1746–1751, 2014.

[23] Barkan, O. Bayesian Neural Word Embedding. AAAI 2017.

[24] Barkan O, Koenigstein N. Item2vec: neural item embedding for collaborative filtering. In IEEE Machine Learning for Signal Processing (MLSP) 2016 Sep 13 (pp. 1-6).

[25] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778).

[26] Huang G, Liu Z, Van Der Maaten L, Weinberger KQ. Densely Connected Convolutional Networks. In CVPR 2017 Jul 21 (Vol. 1, No. 2, p. 3).