Fully Perceptual-Based 3D Spatial Sound Individualization with an Adaptive Variational AutoEncoder

KAZUHIKO YAMAMOTO, The University of Tokyo
TAKEO IGARASHI, The University of Tokyo

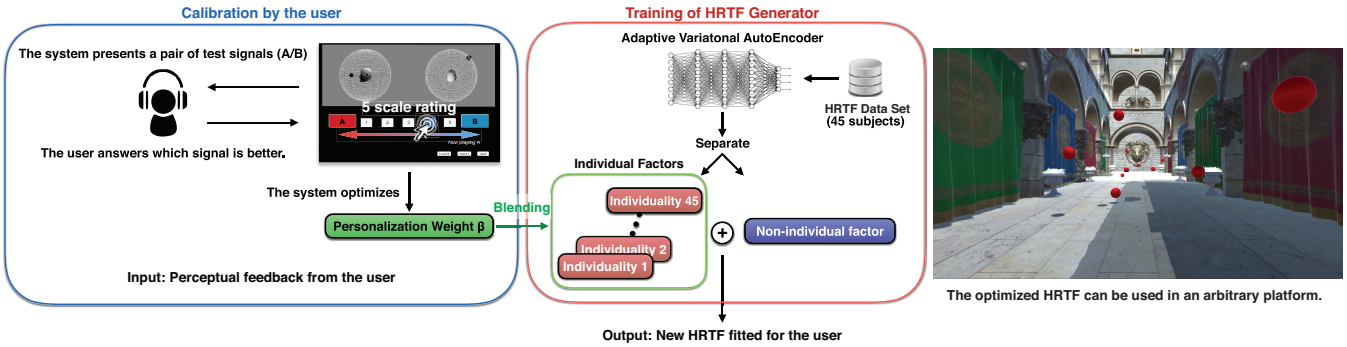

Fig. 1. **The concept of the system**. The user can calibrate their own Human Related Transfer Function (HRTF) for 3d audio spatialization. The system presents a pair of test signals, and the user feedbacks which one is perceptually better. Using this feedback task iteratively, our system optimizes a personalization weight for the user to obtain an individualized HRTF. The personalization weight blends individual factors of HRTF which are extracted from a public HRTF data set during training. After calibration, our system outputs the individualized HRTF in an arbitrary required format for each rendering platform.

To realize 3D spatial sound rendering with a two-channel headphone, one needs head-related transfer functions (HRTFs) tailored for a specific user. However, measurement of HRTFs requires a tedious and expensive procedure. To address this, we propose a fully perceptual-based HRTF fitting method for individual users using machine learning techniques. The user only needs to answer pairwise comparisons of test signals presented by the system during calibration. This reduces the efforts necessary for the user to obtain individualized HRTFs. Technically, we present a novel adaptive variational AutoEncoder with a convolutional neural network. In the training, this AutoEncoder analyzes publicly available HRTFs dataset and identifies factors that depend on the individuality of users in a nonlinear space. In calibration, the AutoEncoder generates high-quality HRTFs fitted to a specific user by blending the factors. We validate the feasibilities of our method through several quantitative experiments and a user study.

CCS Concepts: •**Applied Computing** → **Sound and Music Computing;**

Additional Key Words and Phrases: 3d spatial sound rendering, deep neural network, optimization, sound design in a virtual environment

ACM Reference format:
Kazuhiko Yamamoto and Takeo Igarashi. 2017. Fully Perceptual-Based 3D Spatial Sound Individualization with an Adaptive Variational AutoEncoder. *ACM Trans. Graph.* 36, 6, Article 212 (November 2017), 13 pages.
DOI: 10.1145/3130800.3130838

DOI: 10.1145/3130800.3130838

1 INTRODUCTION

The human auditory system perceives the directions of incoming sounds using both ears. According to the direction from which a sound arrives to the head, an arrival time difference to the left and right ears can be determined. In addition, the sound is intricately diffracted by the shape of the person's head and ears. This diffraction effect depends on the frequency and incoming direction of the sound. Therefore, the spectrums of the sounds that arrive at each ear are modified. We can recognize the localization of the sound by these sound modifications. These two-channel transforms of the spectrums can be represented as finite impulse response filters and are called human-related transfer functions (HRTFs).

Three-dimensional (3D) spatialization of sounds in virtual environments (e.g., VR and games) requires HRTFs to reproduce incoming sounds from various directions using a two-channel headphone. However, HRTFs are highly specific to individuals because they depend considerably on the shape of the user's ears and head. We call the proper HRTFs for individual users as an individualized HRTFs. We know that inappropriate HRTFs can lead to improper localization of the sound source accompanied by an unexpected equalization of the timbre. Such improper localization especially includes front-back and up-down confusions [Middlebrooks 1999; Moller. et al. 1996; Wenzel et al. 1993]. Because of this, we must essentially measure the individualized HRTF for each user. The measurement procedure requires special equipment, including an anechoic chamber, as well as time-consuming and tedious efforts of the user. Thus, using individualized HRTFs for each end user has been impractical. This may explain why 3D sound rendering has not been as popular as visual rendering.

To address these problems, we propose a novel fully perceptual-based optimization of HRTFs for individual users (Figure 1). Our

system requires neither special equipment nor tedious measurement procedures. The user only needs to provide several feedback rating pairwise comparisons of test signals provided by the system based on his or her individual perceptions during calibration. This reduces the user's efforts at obtaining individual HRTFs. Our algorithm uses a novel adaptive variational AutoEncoder [Rezende et al. 2014][Kingma and Welling 2014] trained with a publicly available HRTFs data set. During training, it decomposes HRTFs in the data set into factors based on individual users and the rest. During calibration, our adaptive variational AutoEncoder generates individualized HRTFs for a new user by blending several individualities with personalization weight in nonlinear space. An advantage of this algorithm is that it does not require optimizations for all the spherical directions around the head because the personalization weight is shared within all the directions. Instead, it covers all directions by running optimizations for only a few candidate directions, which has been not addressed in previous studies.

We evaluate our algorithm by several quantitative validations and a user study. The cross validation shows that our algorithm has an ability to generate a fitted HRTF for a new user. In the experiment using synthetic data, our algorithm accurately predicts 3D acoustic field around an obstacle which demonstrates the ability to estimate a new HRTFs. Finally, we show that the individualized HRTFs obtained using our method outperforms best HRTFs in the data set in a user study with 20 users.

The contributions of this study are as follows.

(1) We propose a fully perceptual-based optimization method to obtain individualized HRTFs for users, which reduces user effort of HRTF measurement (Section 4).
(2) We present a novel adaptive variational AutoEncoder model that isolates factors based on individual users during training and synthesizes individualized HRTFs by blending them in a nonlinear space during calibration (Section 5).
(3) We present a hybrid CMA-ES, assisted by a local Gaussian process regression, that accelerates sampling-based optimization of a black box system through user feedback by estimating gradient information (Section 6).

2 RELATED WORK

2.1 Measurement

The straightforward approach to obtaining individual HRTFs involves the actual acoustic measurements in an anechoic chamber. Loud speaker arrays are spherically arranged around the subject's head and two small microphones are inserted into both ears. The subject sits with his or her head placed at the center of these spherical speaker arrays and is instructed to remain still during the long measurement periods. Specific test signals (such as sine sweep signals) is played one by one from the different loud speakers and the signals at the microphones are recorded. By comparing these recordings with those obtained from a microphone placed at the center of the speaker arrays (excluding the subject), the individual HRTF can be computed. Many variants exist for conducting

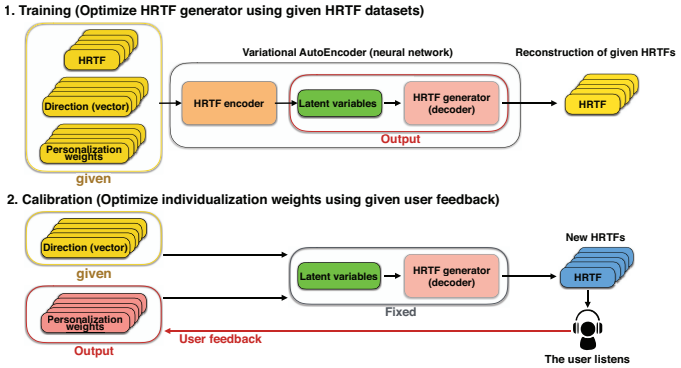

Fig. 2. An algorithm overview. Our algorithm consists of two phases. At the first phase, we train a neural network using a public HRTF data set. At the calibration phase, the system shows test signals generated from the HRTF generator, and the user provides feedbacks according to his/her perceptual direction of the signal. Using this feedback information, the system optimizes personalization weight which is used for the input of HRTF generator to make a new HRTF for the user.

these measurements. However, because such measurements usually require expensive equipment as well as tedious procedures, using these measurements with each end user is impractical and thus prevents their widespread use. An alternative measurement approach is to use reciprocal method [Matsunaga and Hirahara 2010; Zotkin et al. 2006] that much reduces the measurement time. This approach swaps the loud speaker and the microphone positions. It inserts a micro-speaker into the subject's ear and places several microphones around the subject. To measure the HRTF, test signals are played from the inserted speakers and captured by the microphones. However, this has a limitation to capture only the HRTF at a narrow middle range frequency because it highly depends on the specification of such the small loud speaker.

2.2 Numerical Simulation

To avoid actual measurement in an anechoic chamber, many numerical simulation techniques for HRTF have been proposed. These techniques use the scanned 3D mesh of a human's head and ears, and solve an acoustic wave equation to simulate the sound propagation around the head. Two major approaches for solving this acoustic wave equation are the boundary element method (BEM) [Gumerov et al. 2010; Jin et al. 2014; Kahana and Nelson 2007; Katz 2001] and the finite-difference time-domain (FDTD) method [Mokhtari et al. 2008, 2010; Xiao and Liu 2003]. However, these techniques have three critical limitations. First, the simulation usually consumes tens of hours or a few days when using an auditory desktop computer. Second, the user must scan the 3D mesh data, which requires special equipment and additional effort. Third, the scanned mesh of the ears lack the details of the geometry that considerably affects the high frequency domain in an HRTF. To address the first problem, Alok et al. [2014] improved the computational speed of the simulation using an adaptive rectangular decomposition technique (ARD). They achieved a computation time of less than 20 min for a broadband HRTF using auditory machine. However, it still requires the detailed mesh of head and ears which is difficult to obtain.

To avoid the tedious 3D mesh scanning procedure, DeepEarNet [Kaneko et al. 2016] estimates the 3D geometry of a user's ears from RGB photographs using a deep convolutional neural network and then numerically simulate the HRTF using BEM. This system extracts the feature parameters of the ears of the user from the photographs of the ears captured by the user him- or herself from two directions through manual annotations. However, sufficiently capturing the details of the ears in order to estimate the high frequency domain of the HRTF is difficult.

2.3 HRTF Optimization

To obtain fitted HRTF for a user, Zotkin et al. [2004] attempted to select the best matching one from an existing data set. They measured pinna parameters of the user and selected an HRTF of a subject who has the closest pinna parameters. However, this has no guarantee that the picked one is the best HRTF for the user. Several studies have been conducted to optimize an HRTF for an individual user using machine learning techniques. Josef [2014] asked users to adjust the principal component weights (PCW) of the HRTF manually using a slider on a user interface (UI). This was accomplished using listening tests. However, manipulating the unintuitive PCW parameters directly was found to be difficult. Yuancheng et al. [2013a] assumed the "blackbox" human auditory system, which takes a sound cue as input and returns the perceptual direction as a Gaussian process regression model and fits the HRTF for the "virtual" user model using AutoEncoder. Their results imply that nonlinear dimensionality reduction better reconstructs HRTFs than do linear space reduction methods such as principal component analysis (PCA). However, the AutoEncoder they employed, which was trained with the consolidated data of multiple subjects, may spoil the individualities of an HRTF. In addition, their experiments were merely a virtual simulation under rather limited conditions and had not been applied to actual problems.

A major approach to optimizing an HRTF for a new user using machine learning is to solve a regression problem that predicts a low-dimensional reduced HRTF from the anthropometric information of the user's head and ears. Principal Component Analysis (PCA) [Iida et al. 2014] and independent component analysis (ICA) [Huang and Zhuang 2009], which reduce the dimensionality of HRTFs in linear space, have been widely used. Bilinski et al. [2014] represented the sparse vector of a new subject's anthropometric features as a linear superposition of the anthropometric features of a training subset. They then obtained the individual HRTF by using those features as weights to interpolate the HRTF in the training dataset. Various nonlinear regression techniques such as neural networks [Hu et al. 2008] and support vector regression (SVR) [Huang and Fang 2009; Wang and Chan 2013] have also been proposed. However, these regression models cannot adequately express the complex relationships between anthropometric features and low-dimensional HRTFs.

Felipe et al. [2014] proposed a state-of-the-art approach to optimize horizontal plane HRTFs using manifold learning through a nonlinear regression model. This approach uses anthropometric information that is actually measured for each user by professional authors. The dimensionality of the HRTF included in a data set is

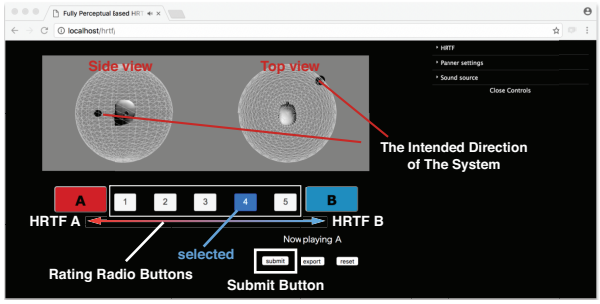

Fig. 3. The user interface pane for gathering the user feedbacks. It runs on web browser. When the user push A (red) or B (blue) button, one of the test signal pair is played. The 3D graphics shows the intended direction of the system from the side and top views. The user rates the pair by 5 pt scale with radio buttons and submit it. Finally, the user exports his/her individualized HRTF data by pressing the export button.

first reduced using IsoMap in a nonlinear space. Next, a regression problem is solved using a neural network. The neural network then processes the anthropometric data of the user and outputs a corresponding reduced (low-dimensional) HRTF. Finally, the system generates the individualized (high-dimensional) HRTF by means of a linear superposition of the neighboring vectors of the reduced HRTF in the IsoMap. However, in addition to the difficulty of measuring the anthropometrics precisely for each user, these anthropometric-based approaches mostly rely on low-dimensional heuristically defined anatomical measures, which are not necessarily sufficient to describe the complicated shape of the ears of humans.

3 ALGORITHM OVERVIEW

Our algorithm consists of two steps (Figure 2): 1) Training of an HRTF generator, which involves learning the individual and non-individual features from an HRTF dataset (§5). 2), Calibration of the HRTF generator, which involves individualizing an HRTF generator for each specific user (§6). In the first step, we train our HRTF generator using an HRTF data set (we used CIPIC data set [Algazi et al. 2001]). The HRTF generator is a generative neural network model that is based on an extension of a conditional variational AutoEncoder [Kingma and P 2014; Sohn et al. 2015]. We extend it by adding 3D convolutional layers (§5.5) designed for HRTF input, as well as novel adaptive layers (§5.6) that separate the individuality and non-individuality factors of the users in a nonlinear space. This neural network takes a set of HRTFs, a continuous vector, and a one-hot vector as input. The continuous and one-hot vectors represent a sampled incoming direction of a sound and subject label (which subject's data are inputted), respectively. It then reconstructs HRTFs by extracting the latent variables as output. After the training of the generative model, our neural network can generate a new HRTF for a given direction around a head using the following three types of input: the latent variables, an intended direction, and a vector defined as personalization weight. Personalization weight represents the amount of contributions from individuals in the dataset in blending.

In the second step (calibration), we optimize the personalization weight to generate an individualized HRTF for the target user. This

step involves interaction with the user, as described in §4. The system optimizes the personalization weight to minimize the difference between the intended spherical direction of the system and the direction perceived by the user. After this calibration, our system can output an arbitrary HRTF format. Most real-time rendering platforms (e.g., game engines) store HRTF data at discrete directions internally and interpolate them at runtime. The system outputs HRTFs at the required directions for each platform and these can be used by an arbitrary rendering scheme depending on each platform.

4 USER INTERFACE

The user obtains individualized HRTFs by running a one-time calibration that roughly consumes 15∼25 min. The calibration application runs on a web browser (Figure 3). The system first presents a pair of test signals and its intended direction. The user then plays the test sound by pressing an A/B selection button. Each of these two test signals is generated from different HRTFs (personalization weights), respectively, and has the same intended direction. We randomly select an audio source from 10 predefined test sounds (e.g., speech, helicopter, short music phrase) and then filter the audio using the generated HRTFs. The intended direction continuously moves spherically around a head and is shown as a moving sphere from side and top views. The user listens to the test signals and provides feedback by selecting one of the 5-scale options that represents the sound that is perceptually closer to the intended direction shown on the screen with "1" meaning that one of two test signals is definitely better, and "5" meaning that the other test signal is definitely better. Thus, "3" means neutral. By iterating this simple pairwise comparisons (approximately 150∼200 times), the system automatically individualizes the HRTFs by optimizing the personalization weight for the target user. The user can stop the calibration at an arbitrary timing (usually when the user satisfied or can not distinguish two test signals). This approach has two advantages. First, obtaining individual HRTFs for users is much easier with this than previous approaches. Second, the effects caused by the acoustic properties of the user's headphones can be considered by using the same headphones for calibration and runtime, something that was not addressed in previous studies.

5 TRAINING WITH PUBLIC HRTF DATA SET

5.1 Data Set

We used the publicly available CIPIC data set [Algazi et al. 2001] which contains HRTFs of both ears actually measured in an anechoic chamber for 45 subjects at 25 azimuths and 50 elevations. In total, it includes 1250 sample directions of HRTF per subject and ear. Each set of data for a direction, subject, and ear is recorded as an impulse response of 200 wave samples with a 44.1kHz sampling rate audio file. Impulse signals are played by a loud speaker array spherically arranged around the head. They are recorded using two small microphones inserted into the ears of each subject. Instead of using the CIPIC angle representation (azimuth and elevation), we redefine the spherical coordinate as yaw and pitch (Figure 4). The yaw θ and pitch angle ψ are measured in a head-centered interaural-polar coordinate system. The yaw is the angle that varies from the back left $-\pi$ to the back right π. The pitch angle varies from the bottom $-\pi/2$ to the top $\pi/2$. We arrange all the sampling points on a unit sphere as 3D vertices and construct a sphere mesh with Delaunay triangulation. This triangulated unit sphere is used for obtaining HRTF data at an arbitrary direction by means of bilinear interpolation. Note that CIPIC dataset has a big hole at the bottom. Because of this, the bilinear interpolation could introduce some artifacts. To alleviate this, one can use arbitrary HRTF interpolation methods [Duraiswaini et al. 2004; Luo et al. 2013b] alternatively.

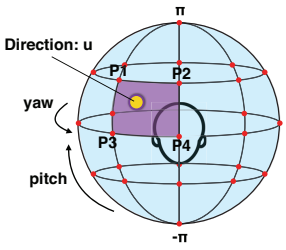

Fig. 4. The representation for an incoming direction of a sound.

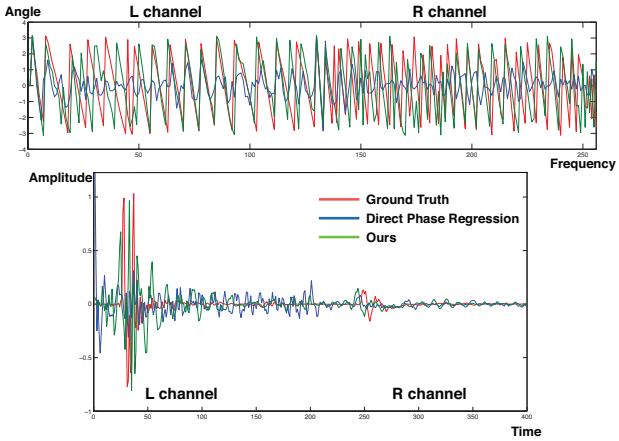

Fig. 5. A comparison of phase (top) and time signal (bottom) reconstruction. Direct phase reconstruction approach (blue) outputs large error that causes unnatural noise in time signal.

5.2 Input Format

During training, our neural network take a sampled direction y (vector), a subject label (one-hot vector) s, and a set of HRTFs x from around the specified direction of the subject as input. It then outputs the reconstructed x' using an adaptive variational AutoEncoder. We train this model to have x' be similar to x. We represent a direction as $y \in \mathbb{R}^{26}$, whose elements are the weights to an overcomplete basis of 26 unit vectors evenly distributed in all directions $\in \mathbb{R}^3$ (red points in Figure 4). For a given direction vector $u \in \mathbb{R}^3$, the system identifies surrounding four unit vectors (P_1, P_2, P_3, P_4), and gives weights (w_1, w_2, w_3, w_4) to them so that $w_1 = s \cdot t$, $w_2 = (1-s) \cdot t$, $w_3 = s \cdot (1-t)$, $w_4 = (1-s) \cdot (1-t)$ where $Yaw_u = s \cdot Yaw_{P_1} + (1-s) \cdot Yaw_{P_2}$, $Pitch_u = t \cdot Pitch_{P_1} + (1-t) \cdot Pitch_{P_3}$. The weights of the other 22 unit vectors are set to zero. We do not use a 3D vector to represent the direction because a neural network tends to be insensitive to the fluctuation of continuous values on a node while being more sensitive to the binary-like activations on each node [Courbariaux and Bengio 2016] The subject label $s \in \mathbb{R}^S$ becomes a one-hot vector that represents the inputted subject data. In our experiment, S became 45 dimensions because the CIPIC data

set includes HRTFs of 45 subjects. If x is the p-th subject's, the corresponding element of s became 1; otherwise, 0.

Our neural network treats HRTF in a spectral domain. The original HRTFs in a time domain can be recovered from both the power spectrum and phase information outputted from the system. For the power spectrum, we compute 256 rectangular-windowed FFTs of each HRTF impulse response and extract only the minimally required power spectrum (128 dimension vectors of LR channels). However, for phase information, we do not use spectral data (angles) directly because the reconstruction error of phase angles are quite sensitive. Actually, previous methods have not solved the regression problem of phase. Alternatively, we solve a rough regression problem of time domain signals, and estimate phase information from them. We use the first 128 samples of the time signal of an HRTF as input to our neural network. In summary, we reconstruct time signals through a neural network, and indirectly estimating only the phase angle (discarding the power spectrum information) from them. Using both the estimated phase and power spectrum, we reconstruct the final HRTFs. This indirect approach to estimating the phase reserves the rough shape of the impulse response, which is difficult to accomplish through direct phase estimation. Figure 5 shows the comparison of phase reconstruction between our method and direct phase estimation. The direct phase regression approach does not reconstruct the phase at all, but rather destroys the shape of the time signal. By contrast, our method successfully simulates the original phase information, thus preserving the time signal shape.

To construct the HRTF input data structure of x, we include not only the impulse response of the HRTF at the exact sampled direction y, but also its several surrounding neighboring impulse responses (Figure 6). In total, we sample 25 directions with 5×5 rectangular grid shapes, in which the center becomes the HRTF at the direction y. The stride of the grid is $\pm 0.08\pi$ rotations for both yaw and pitch on a unit sphere direction from the head. In addition, we obtain the power spectrums and LR time domain impulse responses at each direction using bilinear interpolation in frequency domain on the unit sphere [Langendijk and Bronkhorst 2000; Wenzel and Foster 1993]. To reconstruct an interpolated time signals, we use both interpolated power spectrums and phases. We call this 5×5 grid that stores HRTF information as the *Patch*. This

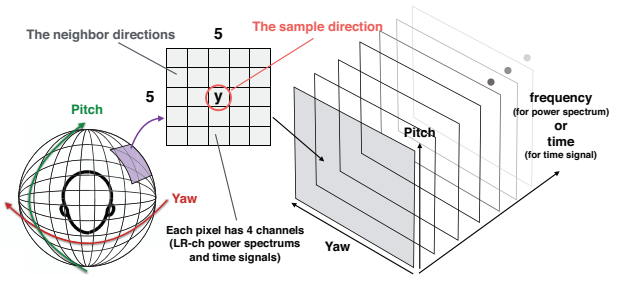

Fig. 6. The input data structure of our neural network (We call HRTF patch). This HRTF patch has voxel like data structure, which encodes spatial correlations of HRTFs. Each voxel has four properties (LR channels of power spectrums and time signals) like color channels of image.

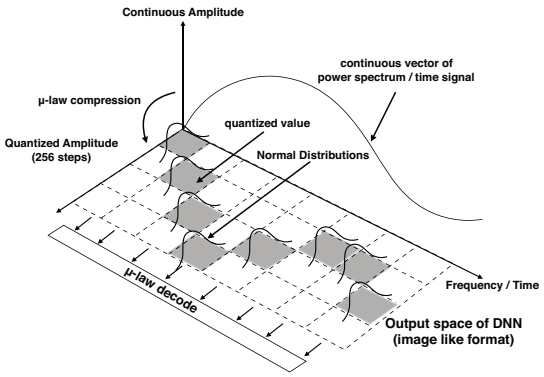

Fig. 7. The output data structure .

patch representation is expected to encode the correlations with surrounding directions. Finally, this input data structure becomes a 3D voxel patch with $5 \times 5 \times 128$ dimensions (128 power spectrums or 128 sample time signals) and each voxel has four color channels (power spectrums and time signals of LR channels) as shown in Figure 6. This becomes the input x of our neural network.

5.3 Output Format

Our neural networkreconstructs the HRTF x' to minimize the difference between the input and output HRTF with an AutoEncoder manner. However, solving a regression problem of a signal that shows a large fluctuation (e.g., time domain audio signal and power spectrum) using a generative neural network is difficult. This is because a neural network smoothes the output throughout the training data. Therefore, the trained result tends to be an "averaged signal," which causes a fatal error. To address this, we use a quantized format similar to WaveNet [van den Oord et al. 2016]. WaveNet predicts time domain audio signals using an image-like quantized format (width: time, height: amplitude), which successfully solves a regression problem of large fluctuated signals. Similarly, we quantize the power spectrums and time signals of HRTFs into 256 steps using μ-law compression. As a result, the output format becomes an image-like representation (Figure 7). Unlike in WaveNet, we do not use one-hot vectors for the final layer nor the SoftMax function. The SoftMax function generalizes all the output of the neural network into [0, 1] probabilities. This is equivalent to solving an unconstrained optimization problem which requires extensive training data. However, the size of our setting's training data is considerably less than that of WaveNet, which can lead to optimization failure. Alternatively, we construct an array of normal distributions on each quantized vector, in which mean values are equal to each quantized value. In addition, we set all the variances to 5 (Figure 7) and minimize the mean squared error of these multiple normal distributions. This addresses the aforementioned problem because it is equivalent to constraining the value range of the solution. To generate a final HRTF, we first compute each quantized value by maximum likelihood estimation from the output and then obtain the result by decoding the quantized values using inverse μ-law compression.

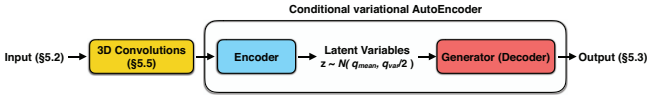

Fig. 8. Our neural network architecture. Our neural network is an extension of a conditional variational AutoEncoder, which reconstructs HRTF from the inputted HRTF through the latent variables.

5.4 DNN Architecture

Figure 8 shows our neural network architecture which has three blocks (please see Appendix B for the detail). Our neural network is an extension of a conditional variational AutoEncoder (Appendix A). Three extensions are used: 1) We introduce 3D convolutions for the input, which are specifically designed for our HRTF patch. 2) We propose an adaptive layer that decomposes the individual and non-individual factors during the training. 3) We introduce residual networks to prevent gradient vanishment. This neural network uses an HRTF patch x, a sample spherical direction y, and the subject label s as input, and reconstructs x' by extracting the latent variables. We divide the HRTF patch x by each channel and input them separately as the power spectrum channels of LR x_{fl} and x_{fr} and the time signals of LR x_{pl} and x_{pr}. Similar to the conventional variational AutoEncoder, the architecture has an encoder and a decoder. The encoder extracts latent variables z_{mean} and z_{var} from the input, and the decoder outputs reconstructed HRTFs in the format described in the previous section from the sampled latent variables z.

5.5 3D Convolutional Layer for HRTF Patch

As shown in Figure 8, we embed 3D convolutional layers for the input of the variational AutoEncoder at the encoder. We expect these convolutions to encode the correlations between the HRTF at the sample direction and its surrounding neighboring directions within an HRTF patch. A typical 3D convolutional layer [Tran et al. 2015] in a neural network shares the filter coefficients of the kernel over the weight tensor. Instead of using this type of layer, we employ a convolutional layer that shares the kernel coefficients only in spatial domains (the yaw and pitch axis) but uses different filters along the frequency axis (Figure 9). This is because the spectral correlation of an HRTF with its surroundings generally has a different structure between the lower and higher frequencies as a result of the frequency dependent diffraction by the subject's head and ears.

We use two convolutional layers for each channel (for four total channels). We set the kernel size of each convolution as 3×3×3 (yaw × pitch × frequency axis), and add zero padding to the frequency axis only. For all directions, we set the stride as 1. Thus, the first convolutional layer transforms each channel of a patch from 5×5×128 to 3×3×128, and the second layer further transforms them to 1×1×128. Note that we did not add bias parameters to this convolutional layer in our experiment.

5.6 Adaptive Layer

Our primary technical contribution is decomposing the individual and non-individual factors from an HRTF dataset during training. This technique uses a novel type of neural network layer called an adaptive layer, which isolates the latent individualities into a tensor

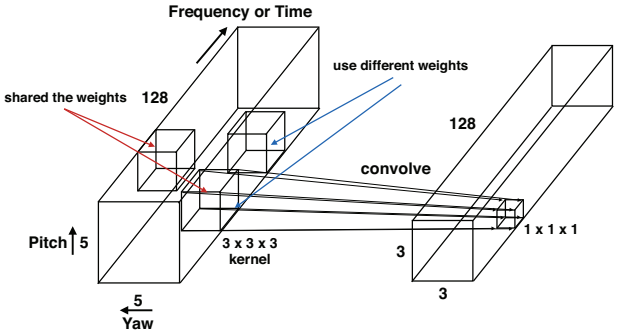

Fig. 9. 3D convolutional layer for HRTF patch.

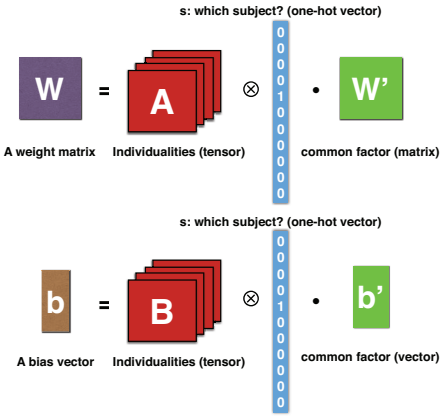

Fig. 10. An adaptive layer that decomposes the function approximation into individual feature and non-individual feature. A one-hot vector s works like switching function depending on the inputted subject's data. $\otimes$ denotes tensor product.

from the weight matrix in an unsupervised manner. In addition to the input vector x, this adaptive layer uses a one-hot vector s during training and a continuous vector β during runtime as input.

Our adaptive layer is based on tensor factorization that employs stochastic gradient descent optimization [Koren et al. 2009]. A common layer in a neural network can be written as a combination of linear and nonlinear functions:

$$y = F(x) = f(Linear(x)) = f(\boldsymbol{W} \cdot x + b), \tag{1}$$

where $x \in \mathbb{R}^M$ and $y \in \mathbb{R}^N$ are the input and output of this layer, respectively. $Linear()$ denotes a linear layer function of the neural network. $\boldsymbol{W} \in \mathbb{R}^{N \times M}$ is a matrix, $b \in \mathbb{R}^M$ is a bias vector, and $f()$ is an arbitrary nonlinear function (e.g., Sigmoid). We decompose this $\boldsymbol{W}$ and b as follows by introducing a new parameter s (Figure 10):

$$y = f(Adapt(x, s)) = f(\boldsymbol{A} \otimes_3 s \cdot \boldsymbol{W'} \cdot x + \boldsymbol{B} \otimes_3 s \cdot b'), \tag{2}$$

where $s = [s_1, ..., s_S]^T$, $s_k \in \{0, 1\}$ is a one-hot vector that represents the subject to which the inputted data belongs. $\boldsymbol{A} \in \mathbb{R}^{N \times M \times S}$, and $\boldsymbol{B} \in \mathbb{R}^{M \times M \times S}$ are tensors. $\boldsymbol{W'} \in \mathbb{R}^{N \times M}$ is a matrix, and $b' \in \mathbb{R}^M$ is a vector. $\otimes_d$ is the dot product between the d-mode expansion of a tensor and vector. We replace the linear layers in the variational AutoEncoder with this adaptive layer, and iteratively input the HRTF data of a randomly selected subject and direction

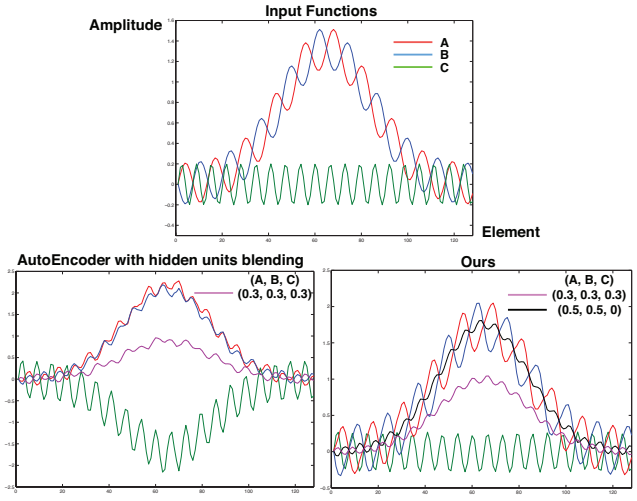

Fig. 11. A comparison between hidden units interpolation approach (bottom-left) and our adaptive layers (bottom-right). We trained two networks with three nonlinear functions A, B and C (Top). Red, blue, green lines at bottom two graphs represent the reconstructed functions respectively. Purple lines denote a blending of three functions equally, and black line denotes a blending of A and B. Hidden units interpolation approach diminishes the details of each function while our method preserves them.

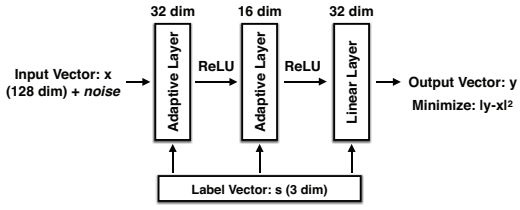

Fig. 12. AutoEncoder network for validation of our adaptive layer.

during optimization. The adaptive layer gradually inserts the individualities of the inputted HRTF patch into the tensor $\boldsymbol{A}$ and $\boldsymbol{B}$ as if the s becomes the switcher with respect to the selected subject. In addition, non-individualities that are shared with all subjects are included in the matrix $\boldsymbol{W}'$ and the vector b' during stochastic optimization.

This adaptive layer allows us to interpolate, emphasize, diminish, and blend each individuality in the trained dataset by adjusting the personalization weight vector $\beta \in \mathbb{R}^S$ at runtime rather than using s as an additional input. To achieve a similar but not necessarily identical objective, several approaches exist that morph data into different categorized data continuously by interpolating several sampled hidden units extracted with AutoEncoder (e.g., using procedural modeling of a 3D mesh [Yumer et al. 2015] and controlling and stylizing the human character motion [Holden et al. 2016]). However, these approaches are limited in terms of their ability to distinguish many nonlinear functions, which is crucial to solving our target problem. Figure 11 shows a comparison of three simple functions morphing between hidden units interpolation approach and our approach after performing the same number of iterations (although this number is unfavorable with our approach). We use a dual-stacked AutoEncoder as shown in Figure 12) for our experiment. (Note that for hidden units interpolation, we replace each adaptive layer with a common fully connected layer.) The hidden units interpolation approach diminishes each feature of the functions. This is crucial to solving our target problem because sharp peaks and dips in the spectral domain are commonly important specifications for an HRTF. However, our adaptive layer successfully enables us to reconstruct the details of each feature and blend them.

In addition, we introduce residual neural networks [He et al. 2015] into the adaptive layers in the encoder (Figure 19) and decoder (Figure 20) in order to reduce the training error of deeper neural networks. A residual network has a shortcut connection as given by the following equation.

$$y = Adapt(x, s) + \boldsymbol{U} \cdot x \tag{3}$$

where $\boldsymbol{U}$ denotes a matrix to project the input x into the output dimension space of y.

6 OPTIMIZING FOR AN INDIVIDUAL USER

After training, we calibrate the HRTF generator (the decoder of the neural network) to obtain an individualized HRTF for a user. To this end, we assume the individuality of the optimized HRTF for an individual user can be expressed as a blending of the trained individualities of HRTFs in the dataset in a nonlinear space. Thus, we now replace the binary subject label s in Eq.(2) with a continuous personalization weight $\beta = [\beta_1, ...\beta_S]^T$ which is called personalization weight. When this β is used, $Adapt(x, s)$ becomes $Adapt(x, \beta)$. Each β_i takes [0,1] continuous value while s is the binary one-hot vector, and is constrained as $\sum_i^S \beta_i = 1$. Finally, the adaptive layer in this phase is reformulated to

$$y = Adapt(x, \beta) = f(\boldsymbol{A} \otimes_3 \beta \cdot \boldsymbol{W}' \cdot x + \boldsymbol{B} \otimes_3 \beta \cdot b'). \tag{4}$$

This representation means the optimized individualization transformation matrices to the user can be expressed as $\boldsymbol{A}\otimes_3\beta$ and $\boldsymbol{B}\otimes_3\beta$, which are blendings of the individualities of the subjects included in the trained data set. Similarly, the latent variables z, which are necessary for generating a new HRTF, are also transformed using β as:

$$\bar{z}_{mean} = \boldsymbol{Z}_{mean}(y) \cdot \beta, \quad \bar{z}_{var} = \boldsymbol{Z}_{mean}(y) \cdot \beta, \tag{5}$$

$$\bar{z} \sim \mathbb{N}(\bar{z}_{mean}, \frac{1}{2}\bar{z}_{var}), \tag{6}$$

where $\boldsymbol{Z}_{mean} \in \mathbb{R}^{L\times S}$, $\boldsymbol{Z}_{var} \in \mathbb{R}^{L\times S}$ are matrices in which each column is the pre-computed latent vector $(z^1_{mean}(y), ...z^S_{mean}(y))$ and $(z^1_{var}(y), ...z^S_{var}(y))$ that correspond to the subject. Furthermore, $z^s_{mean}(y)$ and $z^s_{var}(y)$ are switched by the direction y. Note that $z^s_{mean}(y)$ and $z^s_{var}(y)$ are pre-computed using the trained model for each direction before this step is performed. L denotes the dimensions of the latent variables, and we use 32 for our experiments. We use the blended $\bar{z}$ for the latent variables in the individual feature vector of the user.

The system optimizes the personalization weight vector β for an individual user by fixing the other parameters $\boldsymbol{A}$ and $\boldsymbol{B}$, as well as the matrices $\boldsymbol{W}'$ and bias vectors b'. This approach has the advantage of dramatically reducing the DoFs of the design variables for

optimization purposes because it can eliminate the need for multiple optimization runs when considering all spherical directions. This means optimizing only a blending vector β covers the individualities of the user through all directions.

6.1 Optimizing Personalization Weight through User Feedback

The optimization procedure for the personalization weight β is interactive with the user as described in §4. The user gives relative scores for two individualization weights β_i and β_j. With this input, our optimization problem is reformulated into a minimization problem $\arg\min_\beta Q(\beta)$ where the absolute cost values $Q(\beta)$ are computed from the relative scores as described in a later section. By running this procedure iteratively, the system optimizes the β.

To optimize this black box system, Bayesian optimization [Brochu et al. 2010] is widely used. However, it requires absolute evaluation values at each step, which are not immediately available in our setting. Alternatively, we use a hybrid optimization scheme [Chen et al. 2009] as an evolutional strategy (we use CMA-ES [Hansen et al. 2003]) and a gradient descent approach (we use the BGFS-quasi-Newton method). The optimization procedure is shown in Algorithm 1. We introduce local Gaussian process regression (GPR) [Matheron 1963] to accelerate the optimization. This method estimates the local landscape of the cost function from discrete sampling to obtain the gradients. Figure 13 shows a comparison between with and without using gradient information estimated by GPR. We minimize the EggHolder function for this evaluation using four conditions (N=8, 32). CMA-ES requires N times of function evaluation (pair-wise comparisons) at each iteration. In our target problem, the number of samplings should be small because the sampling size is proportional to the user's effort. However, when the number of samplings N seeded by CMA-ES is few, the optimization without GPR tends to be trapped by a bad local minima. As shown in Figure 13, our technique addresses this problem and converges to a better solution with considerably fewer iterations than in previous methods.

At a single iteration during optimization, we first sample N sets of β samplings $\beta_1, ..., \beta_N$ using common CMA-ES procedure (we used $N = 8$). Let $\mathbb{P}$ be a set of pairs of indices $(1, 2), ..., (N-1, N)$. For each $(i, j) \in \mathbb{P}$, the system generates two test signals $S(\beta_i)$, $S(\beta_j)$, presents the pair side by side to the user, and requests the user to rate each to generate a relative score. After collecting the user feedback for all pairs, the system stores $N/2$ pairs of different βs and their relative scores. After computing the absolute value of the cost q for each β samplings using these relative scores, we estimate the local landscape of the continuous cost function $Q(\beta)$ from the discrete q. We represent $\mathbb{Q} = (q_1, ..., q_N)$ as a set of indices of sampling points. Using this estimated cost function, the system computes the gradients, and updates the covariance matrix in CMA-ES with quasi-Newton. We detail this procedure in the next subsection.

6.2 Estimating the Local Landscape of the Cost Function

The system requests that the user provides feedback regarding the two test signals that correspond to each β pair. The system then

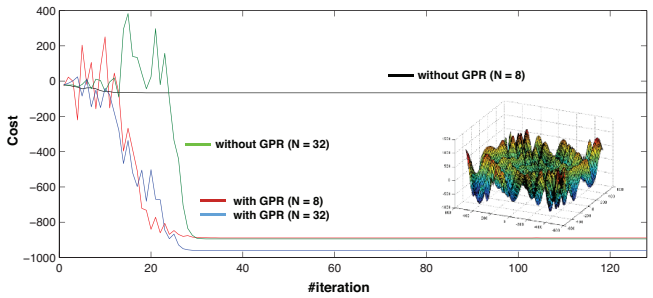

Fig. 13. Comparison of convergence curves between CMA-ES with/without GPR. We use EggHolder function for this evaluation. When the number of samplings seeded by CMA-ES is fewer, the optimization without GPR fails to bad local minima while our technique converges to better solution with much fewer iterations.

ALGORITHM 1: Hybrid CMA-ES assisted by Local GPR

1: **Until convergence**
2: Generate β samplings using Eq.(14) and make P test pairs $\mathbb{P}$.
3: Gathering the user feedbacks for each pair.
4: Computes absolute cost q for each β samplings.
5: Estimating the cost function of each sampling using GPR
6: Sort the samplings by the order of the cost to form the new parent population in CMA-ES
7: The weighted mean $y_w^{(g)}$ is computed from the new parent population .
8: Quasi-Newton updates of $y_w^{(g)}$ using the gradients estimated by GPR.
9: Update the covariance matrix $C^{(g)}$ and global step size in CMA-ES using Chen et al. [2009], respectively.
10: **end**

stores relative scores for the β pairs $\mathbb{P}$. Given these relative scores, we compute the absolute value of the sampling cost q for each sampling β. Our formulation is derived from Koyama et al. [2014], which estimates the consistent goodness field of high dimensional parameters through unreliable crowd sourced rating tasks. Their approach solves a minimization problem with two constraints:

$$\arg\min_q \left(E_{relative}(q) + \omega E_{continuous}(q)\right), \tag{7}$$

where $\omega > 0$ balances the two constraints (we set 5.0). $E_{relative}(q)$ is the relative score-based constraint and is represented as

$$E_{relative}(q) = \sum_{(i,j)\in\mathbb{P}} ||q_i - q_j + d_{i,j}||^2, \tag{8}$$

where $d_{i,j}$ denotes the offset determined by the rating between i-th and j-th samples.

$$d_{i,j} = \begin{cases} 1 & (relative\ score\ =\ 1), \\ 0.5 & (relative\ score\ =\ 2), \\ 0 & (relative\ score\ =\ 3), \\ -0.5 & (relative\ score\ =\ 4), \\ -1 & (relative\ score\ =\ 5). \end{cases} \tag{9}$$

Note that the sign of $d_{i,j}$ is opposite to Koyama et al., because our optimization is a minimization problem. In addition, we enforce the continuity of the cost function by $E_{continuous}(q)$:

$$E_{continuous}(q) = \sum_{i \in \mathbb{Q}} ||q_i - \sum_{i \neq j} (1 - \frac{|\beta_i - \beta_j|}{\sum_{i \neq k} |\beta_i - \beta_k|}) q_j||^2. \quad (10)$$

In this equation, we constrain the absolute costs of two sampling β to become closer when the distance of the two β diminishes. This minimization problem Eq.(8) can be solved as a linear least square problem.

Finally, we estimate the local landscape of the cost function $Q(\beta)$ of the β samplings using multidimensional GPR. We include discrete q samplings obtained by Eq.(8) into GPR. This approximate function can be used to estimate the gradients, which are required by the quasi-Newton method as described in the following paragraph. Note that although GPR is expensive with high dimensions, it is not a serious problem in our case because the dimension of a design parameter would not increase to such a high dimension.

6.3 Optimization

We employ a hybrid optimization scheme [Chen et al. 2009] of an evolutional strategy to minimize $Q(\beta)$ with respect to β. Note that we used CMA-ES [Hansen et al. 2003] and a gradient descent approach (quasi-Newton method.) This hybrid approach first updates the design parameter using gradient information to search for the local optima and to escape from bad local optima. The evolutional strategy aspect generates the offspring (sampling) using two characteristic variation operators, and additive Gaussian mutation alternately.

$$q^g = QuasiNewtonUpdate(z^g), \quad (11)$$

$$z^{(g+1)} = q^{(g)} + \rho^{(g)} \boldsymbol{B}^{(g)} \boldsymbol{D}^{(g)} y^{(g)}, \quad (12)$$

$$\boldsymbol{B}^{(g)} \boldsymbol{D}^{(g)} y^{(g)} \sim \boldsymbol{N}(0, \boldsymbol{C}^{(g)}), \quad (13)$$

where $z^{(g)}$, g and $\rho^{(g)}$ are the design parameters, iteration step and a global step size respectively. $y^{(g)} \sim N(0, \boldsymbol{I})$ are independent realizations of a normally distributed random vector with zero mean and covariance matrix equal to the identity matrix $\boldsymbol{I}$, and $C^{(g)}$ denotes the covariance matrix which is computed using $q^{(g)}$ and $z^{(g)}$ (please see [Chen et al. 2009] for details).

For our initial guess, we first randomly generate the offsprings $y^{(g)}$ within a range [0, 10] by means of Gaussian distribution, and then enforce a constraint $\sum y^{(g)} = 1$ by dividing all the offsprings by $\sum y^{(g)}$. This constraint can be considered as a portion of the user feedback function, and we can optimize CMA-ES with the common range [0,10]. In addition, we assume β has a sparsity because most of optimized HRTF can be represented by a combination of a few HRTFs included in dataset. Thus, the optimizer randomly drop the elements under the average to zero in β offsprings of CMA-ES (we dropped them to 30% probability). We found this constraint reduces training error.

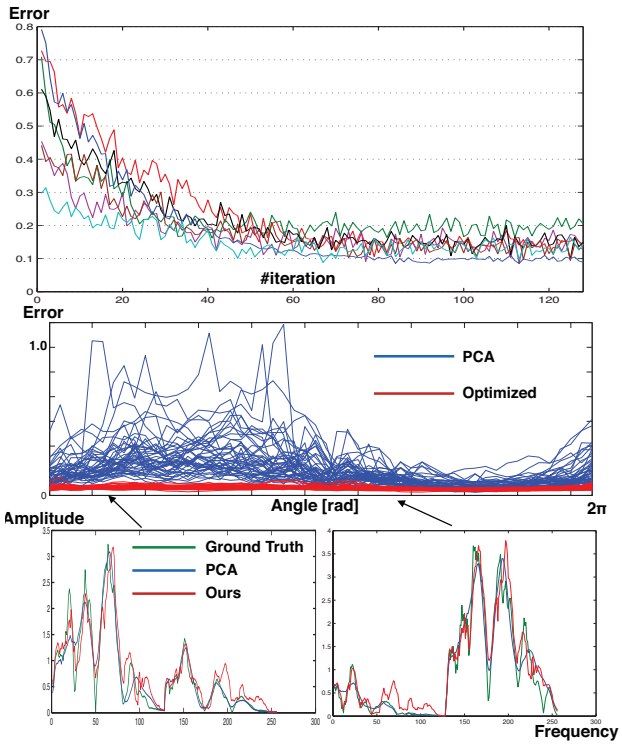

Fig. 14. The result of cross validation. Top: convergence curve of each optimization. Middle: A comparison of estimated errors between PCA and our algorithm on the horizontal plane. Bottom: A comparison between the power spectrum of a target (blue) and optimized result (red).

7 VALIDATION

7.1 Implementation

We implemented our neural network algorithm using C++ (with AVX2 operations) and CUDA from scratch. Our CPU had an Intel Core i7 6900K 3.2 GHz, RAM 128 GB. Our GPU was an NVIDIA Geforce GTX1080x3. The training consumed approximately 8 hours. The GUI application for calibration ran on a web browser. This web application communicates with a GPU server (same machine used for the training) in the background. The calibration algorithm was written in Javascript. The front end GUI application sends βs to the GPU server, and the GPU server generates HRTFs. For our user study, we used the AKG K240 headphones.

7.2 Cross Validation

Confirming the assumption that the individuality of a user can be represented accurately by blending some individualities of other subjects (dataset) is crucial. To confirm this, we applied a cross-validation test to the dataset. After training our neural network with the data of 44/45 subjects, we optimized the personalization weight β to approximate the HRTF data of the remaining subject. In this experiment, we optimized β using a hybrid CMA-ES. For the cost function to minimize, we used a squared mean error of power spectrums and phases between data outputted from the system and the target (the rest subject's data). We conducted this cross validation for all subjects in the CIPIC dataset (45 times). Figure 14 (top)

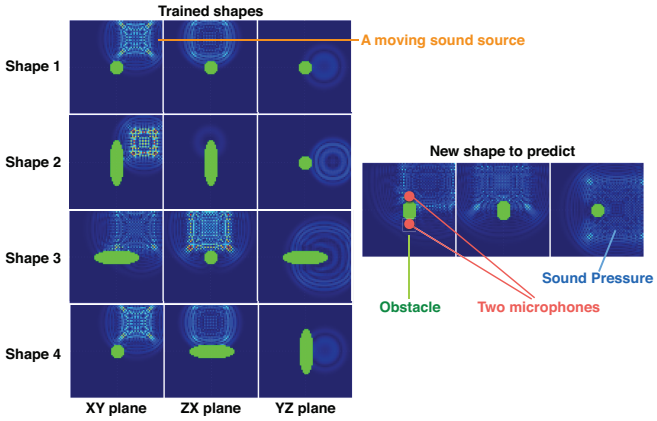

Fig. 15. Synthetic data generation. We conducted 3d acoustic simulation. Green region represents an obstacle.

shows the convergence curves of several optimizations, and Figure 14 (middle) shows a comparison of the prediction error between PCA (32 axis) and our algorithm on the horizontal plane for all the subject. In addition, Figure 14 (bottom) shows power spectrums of a subject at two directions between optimized and target HRTFs. These figures show that all optimizations converge to a similar level of error, and our system can approximate the HRTF of a new user by blending the individualities of the trained dataset. Note that we computed PCA using HRTF at all the directions in this experiment. Naturally, PCA computed for each direction would provide much higher score than this result. However, it requires more than ten thousands of PCA vectors for representing HRTF at all the direction, which is impractical for our target problem.

7.3 Validation with Synthetic Data

We validated the ability of our algorithm to generate appropriate HRTFs by using synthetic data. For synthetic data generation, we simulated an open-space 3D acoustic field around an elliptically shaped obstacle using the finite difference time domain (FDTD) method with a perfectly matched layer [Mokhtari et al. 2008, 2010; Xiao and Liu 2003]. FDTD is an established method used to simulate an impulse response. We set up a three-meter cubic domain with 128×128×128 uniform grid as the simulation field. The simulations were conducted using four shapes as obstacles. Each shape was an oval sphere and had different radii (shape 1: x. 0.2m, y. 0.2m z. 0.2m, shape 2: x. 0.7m, y. 0.2m z. 0.2m, shape 3: x. 0.2m, y. 0.7m z. 0.2m, shape 4: x. 0.2m, y. 0.2m z. 0.7m). We recorded the sound pressure at two opposite sides of each obstacle, modeled on the ears of humans (Figure 15). A sound source was spherically rotated around the obstacle from a position of one meter from the center of the obstacle, and generated an impulse response. We recorded the first 256 samples of sound pressure from the moment each impulse was started. These simulated sounds became virtual HRTFs, and we used them for training data in our algorithm.

We evaluate our method by predicting virtual HRTFs around a new shape obstacle that is not included in the training data. We set the new shape as the intermediate shape between shape 1 and 2 (x. 0.45m, y. 0.2m z. 0.2m). Figure 15 shows a comparison of the errors from the simulated HRTF between our neural network and simple linear interpolation of simulation 1 and 2 on the horizontal plane. To generate this HRTF with our system, we set β=(0.5, 0.5, 0, 0). With all directions, our algorithm had fewer errors than did linear interpolation, which means that our algorithm can generate appropriate HRTFs. Figure 16: bottom shows a comparison of the predicted results of power spectrum at two directions when using both our method and linear interpolation. Apparently, our neural network can estimate specific peaks and dips than can linear interpolation.

7.4 User Study

We employed 20 participants (male:female = 13:7, age: 22~62) and optimized HRTFs for them using our system. The experiment consisted of three steps. In the first step, we investigated the best-fitted CIPIC HRTF for each participant. Here, we conducted a progressive comparison [Takahama et al. 2016] to estimate the best-fitted HRTF. Therefore, a participant compared 44 pairs of HRTFs because 45 CIPIC HRTFs exist. We showed each participant 44 pairs of test sounds convolved by two CIPIC HRTFs using the same GUI in our system (FIgure 3); the participant indicated the best among them. This task consumed 15~20 min. The best CIPIC HRTFs were used in the third step as baselines to evaluate our optimized HRTF. One might question the robustness of the selection of the best CIPIC HRTF. We therefore run an informal pilot study to answer the concern. We requested a participant to find the best HRTF six times. Unfortunately, this was not perfectly repeatable, but we observed the participant selected same HRTF three times in the six explorations. We can consider this small variation in the result is not

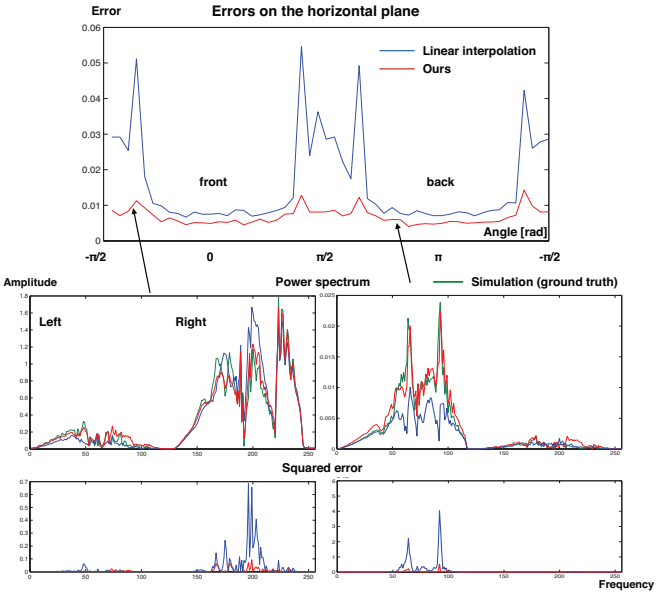

Fig. 16. The result for predicting a new virtual HRTF. Top row shows the errors on the horizontal plane of the obstacle. Bottom shows comparisons of power spectrums between our neural network and simple linear interpolation at two directions.

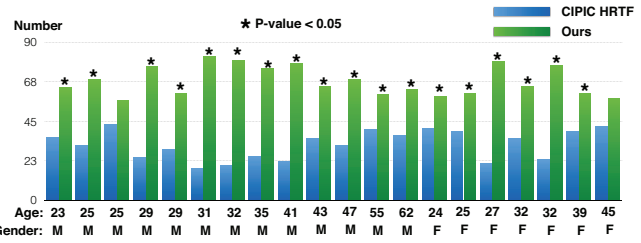

Fig. 17. The result of user study. The third column shows how many numbers of options are selected as better HRTF for each participant between best fitted CIPIC HRTF and optimized HRTF by our system.

a critical problem in our experiment because it means that the selected HRTFs are equally good.

We next requested that the participants calibrate their HRTFs using our system. We ordered each participant to answer at least 100 pairwise comparisons. We did not decide the maximum times of comparisons and when a participant indicated that he or she was satisfied, we stopped the calibration. We measured calibration time and the number of mouse clicks (number of pairwise comparisons) for each participant. The participants used the UI described in §4. The calibration consumed 20~35 minutes for each participant. The number of pairwise comparison was 109~202 times. This calibration time was much shorter than the actual measurement for obtaining fitted HRTFs.

After each calibration, we conducted a blind listening test to compare the HRTFs obtained using our method and the best fitted CIPIC HRTFs. In this step, we showed each participant 100 pairs of test sounds. One of the test sounds in each pair was convolved by an optimized HRTF and the other was convolved by the best CIPIC HRTF for the participant. The test sound to convolve was randomly selected from 10 prepared sounds (e.g., short music, helicopter, and speech) and played 100 times. We requested that each participant use the same GUI as during the calibration to select one test sound from each pair that showed better spatialization . We requested that each participant select only either 1 or 5 from among the option buttons. The order (A or B) in which test sounds were played using optimized HRTF and best CIPIC HRTF for each presented pair was random. We did not inform the participants whether the selection was the optimized HRTF or the best CIPIC HRTF. Figure 17 shows the number of times the better HRTF was selected by each participant. These results show that the optimized HRTFs were significantly better for almost all participants (p-value<0.05 by Chi-squared test for 18/20 subjects) than were the best CIPIC HRTFs, indicating that our system successfully optimizes HRTFs for individual users.

8 LIMITATIONS AND FUTURE WORK

This study presented a fully perceptual-based HRTF optimizer for individual users using a machine learning technique. However, several limitations are remained for future work to overcome.

First, it is not clear how well the dataset we used spans the space of HRTFs. Investigating the variance of HRTFs is an important future work. Second, evaluating the absolute quality of final results after calibration is difficult. To analyze this, more user studies (including the evaluation of azimuth error, elevation error, front-back reversal rate, and externalization percentage) should be conducted in future work. Third, although our approach achieves much faster optimization of HRTFs compared to existing methods, approximately 30 minutes of calibration time is still long. In our user study, we found almost all the optimized individualization weight vectors became very sparse. This means a few elements in the vector are large and the other elements are close to zero. This observation implies that we can decrease the calibration time by reducing the dimensions in the individualization weights that are approaching to zero during the calibration, or using sparse coding techniques for the individualization weights in future work. Finally, our approach for training the individualities would not scale well when the number of subjects in the dataset is much larger. A possible solution is clustering the subjects beforehand (using some features of HRTF like principal component vectors), and reducing the number of "domain" DoFs.

Our adaptive layer could be used in a wide range of other applications (e.g., from mesh morphing to animation generation). However, the following problem remains: training time increases in proportion to different categories of individuality, as the number of training parameters (tensors at each layer) increase. To address this, clustering and reducing the DoFs of the extracted individualities in each adaptive layer should be considered. Recent studies using a Gram matrix at each layer in a DNN for image styling [Gatys et al. 2016] is expected to be useful in solving this problem.

ACKOWLEDGEMENTS

This work was supported by ACT-I, JST.

REFERENCES

V. R. Algazi, R. O. Duda, D. M. Thompson, and C. Avendano. 2001. The CIPIC HRTF Database. In *IEEE Workshop on Applications of Signal Processing to Audio and Electroacoustics*. 99–102.

P. Bilinski, J. Ahrens, M. Thomas, I. Tashev, and J. Platt. 2014. HRTF magnitude synthesis via sparse representation of anthropometric features. In *Proc. IEEE Int. Conf. Acoust., Speech, Signal Proces.*

Eric Brochu, Tyson Brochu, and Nando de Freitas. 2010. A Bayesian Interactive Optimization Approach to Procedural Animation Design. In *Proc. of ACM SCA*. 103–112.

Xuefeng Chen, Xiabi Liu, and Yunde Jia. 2009. Combining Evolution Strategy and Gradient Descent Method for Discriminative Learning of Bayesian Classifiers. In *Proc. of Genetic and Evolutionary Computation*. 507–514.

Djork-Arne Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *Proc. of ICLR*.

Matthieu Courbariaux and Yoshua Bengio. 2016. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or − 1. In *arXiv*.

R. Duraiswaini, D.N. Zotkin, and N.A. Gumerov. 2004. Interpolation and range extrapolation of HRTFs [head related transfer functions]. In *ICASSP*.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethges. 2016. Image Style Transfer Using Convolutional Neural Networks. In *Proc. of IEEE CVPR*.

Felipe Grijalva, Luiz Martini, Siome Goldenstein, and Dinei Florencio. 2014. Anthropometric-Based Customization of Head-Related Transfer Functions using Isomap in The Horizontal Plane. In *ICASSP*.

Nail A. Gumerov, Adam E. O ' Donovan, Ramani Duraiswami, and Dmitry N. Zotkin. 2010. Computation of the head-related transfer function via the fast multipole accelerated boundary element method and its spherical harmonic representation. In *J. Acoust Soc. Am*, Vol. 127.

N Hansen, SD Muller, and P Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). In *Evolutionary Computation*. 1–18.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. In *Proc. of CVPR*.

Daniel Holden, Jun Saito, and Taku Komura. 2016. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Transaction on Graphics (SIGGRAPH)*, 35, 4 (2016), 138:1–138:11.

Josef Holzl. 2014. A Global Model for HRTF Individualization by Adjustment of Principal Component Weights. In *Diploma Thesis.*
Hongmei Hu, Lin Zhou, Hao Ma, and Zhenyang Wu. 2008. HRTF personalization based on artificial neural net- work in individual virtual auditory space. In *Applied Acoustics*, Vol. 69. 163–172.
Q Huang and Y Fang. 2009. Modeling personalized head- related impulse response using support vector regressions. In *J. Shanghai Univ.*
Q. Huang and Q. Zhuang. 2009. HRIR personalisation using support vector regression in independent feature space. In *Electron. Letter*, Vol. 45.
PK. Iida, Y. Ishii, , and S. Nishioka. 2014. Personalization of head-related transfer functions in the median plane based on the anthropometry of the listener ' s pinnae. In *J. Acoust Soc. Am.*
Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proc. of ICML.*
Craig T. Jin, Pierre Guillon, Nicolas Epain, Reza Zolfaghari, Andre van Schaik, Anthony I. Tew, Carl Hetherington, and Jonathan Thorpe. 2014. Creating the Sydney York Morphological and Acoustic Recordings of Ears Database. In *IEEE Transactions on Multimedia*, Vol. 16.
Y. Kahana and P. A. Nelson. 2007. Boundary element simulations of the transfer function of human heads and baffled pinnae using accurate geometric model. In *Journal of sound and vibration.* 552–579.
Shoken Kaneko, Tsukasa Suenaga, and Satoshi Sekine. 2016. DeepEarNet: individualizing spatial audio with photography, ear shape modeling, and neural networks. In *AES Conference on Audio for Virtual and Augmented Reality.*
B. F. Katz. 2001. Boundary element method calculation of individual head-related transfer function. i. rigid model calculation. In *J. Acoust Soc. Am.*
Kingma and Diederik P. 2014. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems.*
D Kingma and J P Ba. 2014. Adam: A method for stochastic optimization. In *CoRR abs/1412.6980.*
Diederik P Kingma and Max Welling. 2014. Auto-encoding variational Bayes. In *Proc. of ICLR.*
Yehuda Koren, Rovert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. In *IEEE Computer,*, Vol. 42. IEEE, 30–37.
Yuki Koyama, Daisuke Sakamoto, and Takeo Igarashi. 2014. Crowd-powered parameter analysis for visual design exploration. In *Proc. of ACM UIST.* 65–74.
E.H.A. Langendijk and A.W. Bronkhorst. 2000. Fidelity of three-dimensional-sound reproduction using a virtual auditory display. In *J. Acoust. Soc. Am.*
Yuancheng Luo, Dmitry N. Zotkin, Hal Daume, and Ramani Duraiswami. 2013b. Kernel regression for Head-Related Transfer Function interpolation and spectral extrema extraction. In *ICASSP.*
Yuancheng Luo, Dmitry N. Zotkin, and Ramani Duraiswami. 2013a. Virtual AutoEncoder Based Recommendation System for Individualizing Head-Related Transfer Functions. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics.*
G Matheron. 1963. Principles of geostatistics. In *Economic Geology.* 1246–1266.
Noriyuki Matsunaga and Tatsuya Hirahara. 2010. Reexamination of fast head-related transfer function measurement by reciprocal method. In *J. Acoust Soc. Ja*, Vol. 31, 6.
Alok Meshram, Ravish Mehra, and Dinesh Manocha. 2014. Efficient HRTF Computation using Adaptive Rectangular Decomposition. In *AES 55th International Conference.*
J.C Middlebrooks. 1999. Virtual localization improved by scaling non-individualized external-ear transfer functions in frequency. In *J. Acoust. Soc. Am. 106.*
P. Mokhtari, H Takemoto, R. Nishimura, and H. Kato. 2008. Computer simulation of hrtfs for personalization of 3d audio. In *In Universal Communication, IEEE. ISUC ' 08. Second International Symposium.* 435–440.
P. Mokhtari, H Takemoto, R. Nishimura, and H. Kato. 2010. Computer simulation of kemar ' s head-related transfer functions: verification with measurements and acoustic effects of modifying head shape and pinna concavity. In *Principles and Applications of Spatial Hearing.* 179–194.
H. Moller., M.F. Sorensen., Jensen C.B, and HammershOi. 1996. Binaural technique: do we need individual recordings?. In *J. Audio Eng. Soc. 44, 451e469.*
Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic backpropagation and approximate inference in deep generative models. In *Proc. of ICML.*
Kihyuk Sohn, Honglak Lee, and Xinchen Yan. 2015. Learning Structured Output Representation using Deep Conditional Generative Models. In *Advances in Neural Information Processing Systems.*
Ryusuke Takahama, Toshihiro Kamishima, and Hisashi Kashima. 2016. Progressive Comparison for Ranking Estimation. In *Proc. of IJCAI.*
Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. 2015. Learning Spatiotemporal Features with 3D Convolutional Networks. In *Proc. of IEEE ICCV.*
Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Ko-ray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. In *CoRR abs/1609.03499.*
Z. Wang and C. F. Chan. 2013. HRIR customization using common factor decomposition and joint support vector regression. In *Eur. Signal Process. Conf.*
E.M Wenzel, D. J Arruda, and D.J Kistler. 1993. Localization using non-individualized head-related transfer functions. In *J. Acoust. Soc. Am. 94.*
E.M Wenzel and S.H Foster. 1993. Perceptual consequences of interpolating head-related transfer functions during spatial synthesis. In *Proc. of Workshop on Applications of Signal Processing to Audio and Acoustics.*
T. Xiao and Q. H. Liu. 2003. Finite difference computation of head-related transfer function for human hearing. In *J. Acoust Soc. Am.*
M. E Yumer, P Asente, R Mech, and L. B Kara. 2015. Procedural Modeling Using Autoencoder Networks. In *Proc. of ACM UIST.* ACM.
D. N. Zotkin, R. Duraiswami, and L. S. Davis. 2004. Rendering localized spatial audio in a virtual auditory space. In *IEEE Transactions on Multimedia, vol. 6(4).*
Dmitry N. Zotkin, Ramani Duraiswami, Elena Grassi, and Nail A. Gumerov. 2006. Fast head-related transfer function measurement via reciprocity. In *J. Acoust Soc. Am*, Vol. 120.

A VARIATIONAL AUTOENCODER

Our network architecture is based on variational AutoEncoder [Kingma and Welling 2014; Rezende et al. 2014] Variational AutoEncoder is a generative model of a deep neural network. We used conditional variational AutoEncoder [Kingma and P 2014; Sohn et al. 2015]. It consists of a decoder $p_\theta(x, y|z)$ and the variational posterior encoder $q_\phi(z|x, y)$, where x, y, and $z$are input, description label, and latent variable respectively, and produces the parameters of each distribution after a series of non-linear transformations. Both the model (θ) and variational (ϕ) parameters will be jointly optimized with stochastic gradient variational Bayes (SGVB) algorithm according to a lower bound on the log-likelihood. This parametrization allows us to capture most of the salient information of x and y in the embedding z. By choosing a Gaussian posterior $q_\phi(z|x, y)$ and standard isotropic Gaussian prior $p(z) \sim N(0, I)$ we can obtain the following lower bound.

$$\begin{aligned} log\ p_\theta(y|x) \quad &= \quad -KL(q_\phi(z|x)||p_\theta(z|x)) \\ &\quad +\mathbb{E}_{q_\phi(z|x)}\left[-log\ q_\phi(z|x) + log\ p_\theta(x, z)\right] \\ \geq -KL(q_\phi(z|x, y)||p_\theta(z)) &+ \mathbb{E}_{q_\phi}(z|x, y)\left[log\ p_\theta(y|x, z)\right], \end{aligned} \tag{14}$$

and the empirical lower bound is written as

$$log\ p_\theta(y|x) \geq -KL(q_\phi(z|x, y)||p_\theta(z)) + \frac{1}{L}\sum_{l=1}^{L} log\ p_\theta(y|x, z^{(l)}), \tag{15}$$

where $KL()$ denotes Kullback-Leibler divergence. Finally, the total loss to minimize can be formulated as

$$L = \quad |x' - x|^2 - \tfrac{1}{2} Mean\left(\sum(1 + z_{var} - z_{mean}^2 - e^{z_{var}})\right), \tag{16}$$

where $Mean()$ represents the mean average. Note that we use only the HRTF data at the sample direction (the center position in a patch although we sampled 5×5 directions for input) for original input x becomes a 512 dimensions vector.

B DNN ARCHITECTURE

Figure 18, Figure 19 and Figure 20 show each block of our neural network. In these figures, red arrows denotes adaptive layers described in §5.6, and blue arrows denotes common linear layer. We divide the HRTF patch x by each channel and input them separately as the power spectrum channels of LR x_{fl} and x_{fr} and the time signals of LR x_{pl} and x_{pr}. Similar to the conventional variational

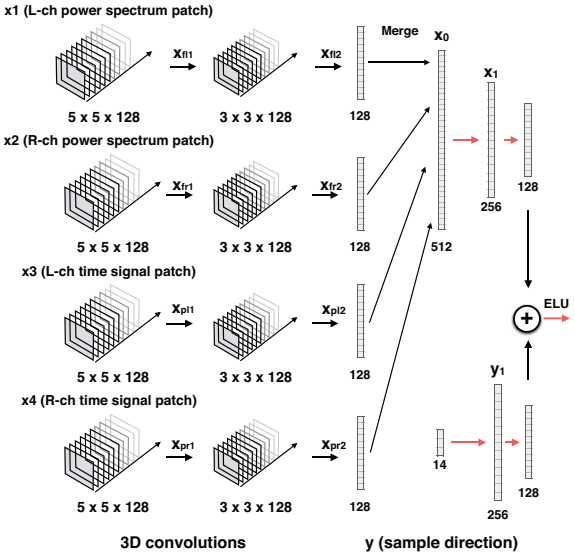

Fig. 18. 3D convolutions block architecture.

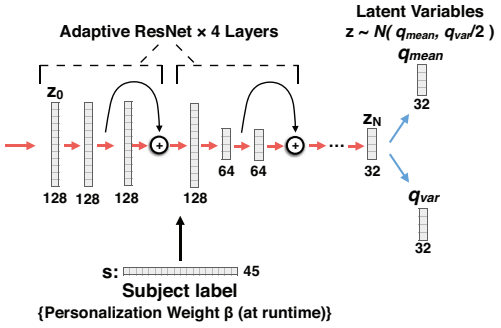

Fig. 19. The encoder block extracts latent variables from input.

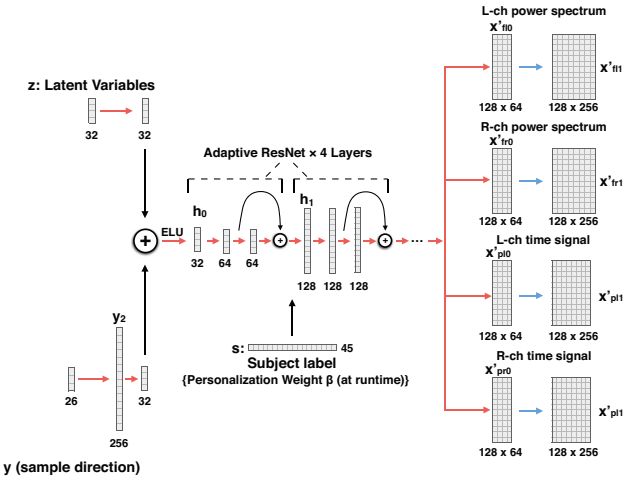

Fig. 20. The decoder block generates reconstructed HRTF from latent variables.

AutoEncoder, the architecture has an encoder (Figure 19) and a decoder (Figure 20). The encoder extracts latent variables from the input, and the decoder outputs reconstructed HRTFs in the format described in the previous section from the sampled latent variables.

Figure 21 shows the equations, where $ELU()$ denotes an exponential linear unit [Clevert et al. 2016]. $Conv()$ is the 3D HRTF patch convolution and $Adapt()$ represents our adaptive layer (which we describe in a later section). The middle of the network has feed forward connections that represent residual networks. $Merge()$ joints the four channels of vectors into a single vector. At the encoder, the system first decomposes the HRTF patch into four channels (left power spectrum x_{fl}, right power spectrum x_{fr}, left time signal x_{pl}, and right time signal x_{pr}), and inputs each channel into independent convolutional layers. After applying the convolutions, the system merges the four channels into a single vector as x_0 and transforms it into x_1 by applying an adaptive layer in order to reduce the number of dimensions. This thus becomes an input vector of the variational AutoEncoder. Using these two vectors, which represent a sample direction and subject label, respectively, as well as the input vector after the convolutions, the encoder of our variational AutoEncoder generates the mean z_{mean} and variation z_{var} vectors of a Gaussian distribution (latent variables in Figure 20). The latent variables can be generated from this Gaussian distribution $z_p \sim \mathbb{N}(z_{mean}, \frac{1}{2}z_{var})$. At the decoder, the system uses the two vectors y and s that match the encoder and latent variables z_p, and reconstructs the center HRTFs of the input HRTF patches of the four channels (L-ch power spectrum, R-ch power spectrum, L-ch time signal, and R-ch time signal) through residual adaptive network layers.

For optimization, we use the mini-batch Adam algorithm [Kingma and Ba 2014] with mini-batch size 16. We set the numbers of the layer N as 4. In addition, we insert the batch normalization layers [Ioffe and Szegedy 2015] before all nonlinear units (ELU function).

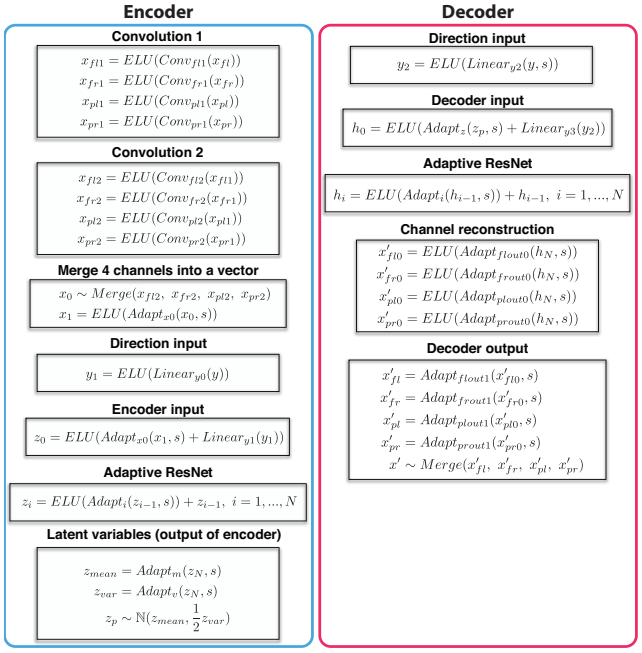

Fig. 21. The equations of our DNN.