vBeats: A Framework for Converting Gestures into Bass Strokes from Indian Tabla using Smartphones

Ayudh Saxena[§], Soumyajit Chatterjee[§], Sandip Chakraborty
Department of CSE, IIT Kharagpur, India

***Abstract*—Human gestures while playing instruments are highly critical. A small incorrect gesture can result in undesirable noise, disturbing the overall synchronization of the orchestra. Certainly, the most obvious way of honing the skills in playing instruments can be accomplished with dedicated practice, albeit the learner must always possess the instrument. In this paper, we propose an initial version of the framework *vBeats* that allows learners to use the COTS smartphones to practice sophisticated percussion-based instruments like tabla without physically having it. Preliminary results on an in-house dataset, with offline inferencing, show that *vBeats* can be used to generate similar waveforms; however, it may need accurate fine-tuning, which we discuss as a part of future work.**

***Index Terms*—multimodal sensing, acoustic signal processing, gesture recognition**

I. Introduction

For ages, music has been a major source of entertainment, and technological advancement has contributed heavily to how humans produce and perceive music. Interestingly, in the last few years, especially during the pandemic, many people across the globe explored the idea of learning how to play different musical instruments mostly to spend their leisure time and to reduce stress [1]. However, one of the primary necessities that limit this self-learning is the requirement of the intended instrument. Subsequently, different solutions have emerged, ranging from simple applications that provide a simulated software version of the instruments like guitars [2] to sophisticated systems like [3], [4], which can act as a proxy when the actual instrument is unavailable.

However, most of these applications often give a superficial experience, and gadgets like [5] are even more challenging to obtain than the actual instrument. This paper tries to mitigate these challenges by introducing the preliminary version of the framework *vBeats*, which tries to generate complex bass strokes of an Indian musical instrument *Tabla* from the in-air gestures captured using COTS smartphones. Drawing motivations from works like [6], *vBeats* in the backend tries to perform a complex cross-modal mapping between the acoustic signatures and the inertial signatures captured from the embedded sensors of the smartphone (Section II). A principled evaluation (Section VI) on an in-house collected dataset (Section V) shows that in runtime *vBeats* can indeed generate the bass strokes with significant similarity. Furthermore, we also identify and discuss the challenges of developing such a system and propose specific ideas for future directions. The current version of *vBeats* with offline inferencing is publicly available at https://github.com/ayudhsaxena/vBeats-implementation.

[§]The authors were students at IIT Kharagpur when this work was completed.

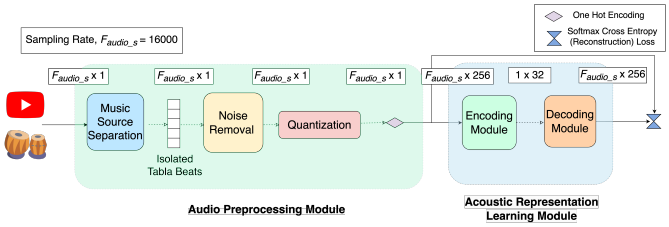


Fig. 1: Acoustic Representation Module

II. Developing *vBeats*: Learning Robust Representations

A. Brief Overview

The crux of *vBeats* relies on the efficient mapping of gestures to acoustic patterns. However, both of these modalities are highly different in their physical properties, and thus a straightforward mapping from one to the other is impossible. For example, one significant change across these two modalities can be observed from the sampling rates with which both are captured. Typically, the gestures captured using inertial sensors are sampled at much lower sampling rates (in the range of $50-100$Hz). In contrast, the acoustic signatures are recorded at significantly higher sampling rates ($16-44.1$kHz). Due to such significant differences in the nature of these two signatures, *vBeats* first extracts the robust representations for each of these modalities and then intelligently uses these representations to learn the mappings across the modalities. The details of the steps follow.

B. Acoustic Representation Module (ARM)

Learning efficient feature representations of acoustic data generated from tabla in real-time scenarios can be extremely challenging due to two major reasons – (a) the presence of noise and additional signatures can highly collude the process and (b) the dearth of publicly available datasets that contain only tabla audio to train the representation learning module. We tackle both challenges by first designing an effective noise removal and quantization module followed by an intelligent choice of using convolutional autoencoders to learn robust feature representations (see Fig. 1). The details follow.

1. Audio Preprocessing To learn robust acoustic representations, we needed a considerable volume of audio data

to train the autoencoder. Naturally, it was not feasible to collect all the audio data in-house. Understanding this, we relied on the publicly available Groove Dataset[7] for training the autoencoder. However, for a robust representation of the acoustic signatures generated by tabla, we need further fine-tuning. For this, we collected ≈ 5.08 hours of tabla audio from YouTube. Notably, the collected audio often contained additional signatures from other instruments and vocals which needed cleaning. We performed this curation using Spleeter [8] and separated the tabla audio from the rest.[1].

Once we obtain the preprocessed audio files containing tabla audio only, we split them into windows of 1sec and apply noise removal. More specifically, we use the SOTA 'noisereduce' [9] library, which takes the windowed audio data sampled at $F_{audio_s} = 16$kHz. Additionally, we also trim any trailing silence with a threshold of 10dB. Finally, once the noise-removed audio is obtained, we further process it for quantization. The primary intuition behind quantizing the audio is to make it more realistic for the model to identify the beats accurately. To achieve this, we first apply a μ-law commanding transformation [10] to the data and then quantize it to 256 possible values. This non-linear quantization produces a significantly better reconstruction than a simple linear quantization scheme. The quantized vectors are then one-hot encoded into the possible 256 values. Thus the output after quantization is of the shape $F_{audio_s} \times 256$ vector, which is then used for robust feature extraction.

2. Acoustic Representation Learning For learning robust acoustic features, we use convolutional autoencoders. The encoder consists of a sequence of groups of 1D convolutional layers, ResNet1D [11] blocks, and 1D max-pooling layers while in the decoder network consists of 1D transposed convolutional layers, ResNet1D blocks, and 1D up-sampling layers. All ResNet1D blocks, across both networks, use the *LeakyReLU* activation function ($\alpha = 0.4$).

For the encoder network we use 15 1D convolutional layers, with each ResNet1D block contributing two convolutional layers and five max-pooling layers with varying sizes. For all the convolutional layers we use 32 kernel filters of size 64. Notably, we use causal convolutions to ensure that the model does not violate the ordering in which we model the data. Finally, we fix the last layer as a dense linear layer of size 32 with sigmoid activation.

The decoder network is an exact mirror image of the encoder network with transposed convolutional layers and up-sampling layers[2]. The objective of the decoder is to reconstruct the input from the latent embedding coming from the encoder. Since we quantized the raw audio to 256 levels, we use softmax on the final logits obtained from the decoder and cross-entropy loss for reconstruction. We train the autoencoder train with a mini-batch size of 48 for 200 epochs (patience = 50) using Adam optimizer (lr $= 0.003$).

[1]Out of the 4 configurations (vocals, drums, bass, and others) we choose drum for this.

[2]We do not use causal convolutions here.

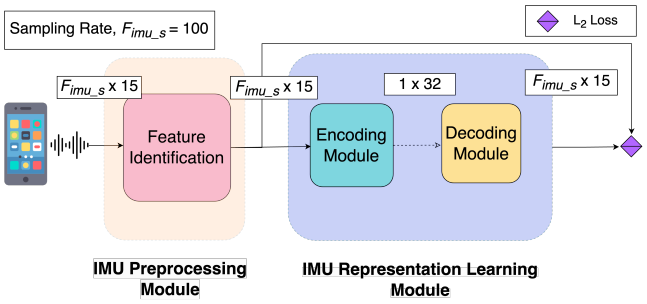


Fig. 2: Gesture Representation Module

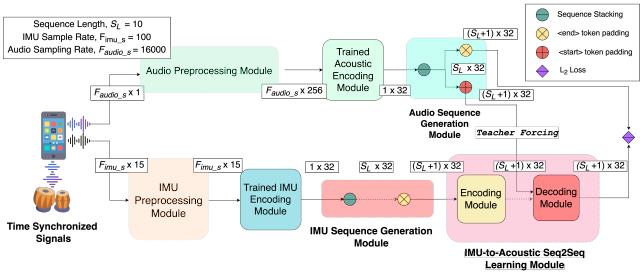


Fig. 3: *vBeats*: Training Regime

C. Gesture Representation Module (GRM)

Similar to learning robust acoustic representations, we must also learn robust features for correctly recognizing gestures. Therefore, we also train a gesture representation learning module as follows (broad overview in Fig. 2).

1. IMU Preprocessing For preprocessing the IMU, we only perform feature identification. We use the IMU signals obtained from the tri-axial accelerometer, gravity, linear accelerometer, gyroscope, and magnetometer sensors available on COTS smartphones. All these signals are recorded at $F_{imu_s} = 100$Hz and then later sampled in a fixed-width sliding window of 1sec with no overlap. Thus the output shape from our preprocessing module becomes $F_{imu_s} \times 15$.

2. IMU Representation Learning Unlike in ARM, we use a shallower convolutional autoencoder for the IMU representation learning module with a similar architecture to the acoustic autoencoder. Here both the encoder and decoder networks consist of 9 1D convolutional layers and 3 max-pooling layers of varying sizes. All convolutional layers have 32 kernel filters of size 5. The encoder submodule's last layer is a linear dense layer with 32 outputs units. Like in ARM, the decoder network is an exact mirror image of the encoder network with the convolutional layers replaced by their transpose and the max-pooling layers replaced with up-sampling layers. For training the IMU autoencoder, we use a mini-batch size of 64 and train it for 100 epochs with Adam optimizer (lr $= 0.006$) and L_2-norm as the loss function.

III. *vBeats*: Training

Once the ARM and GRM modules are trained, we next train the cross-modal mapping model. Motivated by the works like

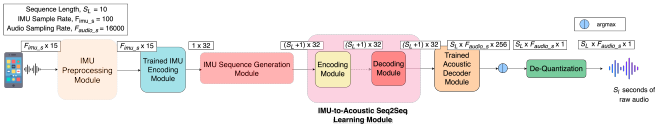


Fig. 4: *vBeats*: During Inference. Current version only performs an offline inference using the gestures recorded by the IMU sensor of a smartphone.

(a) (b)

Fig. 5: Data collection setup – (a) COTS smartphone attached to the hand and (b) Person playing tabla with the smartphone tied on the hand.

IMU2Doppler [6], we train a *seq2seq* model to learn this mapping across the modalities. The designed seq2seq uses LSTM for both the encoder and decoder. The input to the model is time-synchronized IMU and audio signals, which are then independently processed and encoded as mentioned above. The audio signals processed using the trained encoder from the ARM are fed into the audio sequence generation module. In this module, the encoded vectors undergo a concatenation operation to get a sequence of length $S_L = 10$, resulting in vectors of shape $S_L \times 32$. The output from here is divided into two streams – (i) the first stream where the vector is padded with the '⟨start⟩' token, resulting in a $(S_L + 1) \times 32$ vector which is then fed into the decoder of the seq2seq module and (ii) the second stream, where the vector is padded with the '⟨end⟩' token, resulting in a $(S_L + 1) \times 32$ vector, used for calculating the loss[3].

Similarly, the encoded IMU signal is fed into the sequence generation module, which undergoes a concatenation operation to get a sequence of length $S_L = 10$, resulting in vectors of shape $S_L \times 32$. Here we pad the sequence with the '⟨end⟩' token and provide the final sequence as input to the encoder of the seq2seq model. The output of this model is finally passed through a sigmoid activation layer.

A primary concern during training of this seq2seq model was having a minimal volume of training data (see Section V). Thus to avoid overfitting, we used a dropout with a probability value of 0.8 in both our encoder and decoder LSTMs and trained our LSTM encoder-decoder network with a minimum batch size of 4 for 100 epochs using the same optimizer configurations and L_2 Norm as the loss function.

IV. *vBeats*: Generating Tabla Beats from Gestures

During the inference, *vBeats* takes an input from the IMU sensors (embedded in the smartphone) of shape $F_{imu_s} \times 15$ capturing the in-air gestures made by the user. This input is then passed through the trained encoder of the GRM, which encodes it into a vector of shape 32. This encoding is then fed into the IMU sequence generation module the output of which is finally fed into the trained seq2seq model for producing the corresponding acoustic representations of shape $(S_L+1)\times 32$.

After trimming off the last latent embedding of this sequence, assuming it to be the ⟨end⟩ token we finally obtain the intended shape of $S_L \times 32$. This representation is then fed to the trained decoder from the ARM followed by the dequantization module to finally generate the corresponding beats from the gestures. A broad overview of training and inference regimes of *vBeats* is shown in Fig. 3 and Fig. 4.

V. Data Collection

For the in-house dataset collection, we recruited 4 professionally trained participants to play tabla for a total duration of 1.38 hours. We asked each participant to play tabla while a COTS smartphone was tied to the back of the hand[4], such that the length of the phone is perpendicular to the length of the arm as shown in Fig. 5. For capturing the time synchronized data the smartphone was preinstalled with two Android applications for capturing the IMU signatures (at 100Hz) and the acoustic data (at 44.1kHZ).

VI. Preliminary Evaluation

In this section, we discuss the evaluation of *vBeats* using the in-house collected dataset (see Section V). The details follow.

1. Experimental Setup and Metric For the primary performance evaluation of *vBeats* we first split the entire dataset into 90% training and use the 10% held-out dataset for the evaluation. For the evaluation, we obtain the cosine similarity between the *Mel-Frequency Cepstral Coefficients* (MFCC) of the ground-truth audio samples and the audio samples generated by *vBeats* using the recorded inertial signatures for the corresponding time window.

2. Performance Evaluation of *vBeats* Following the aforementioned evaluation setup, we evaluate the performance of *vBeats* using cosine similarity as a metric. Notably, we measure this across different overlapping window lengths. Here the overlaps correspond to the different windows across the IMU data taken as input during runtime. As shown in Fig. 6, we can see that irrespective of the overlapping window sizes *vBeats* generates acoustic patterns that are highly similar to the ground-truth acoustic signatures. However, despite such a high similarity in the MFCC coefficients, we observe considerable noise in the generated audio. Understanding this concern, we further investigate the output audio as follows.

3. Quality of Generated Acoustic Signatures From a deeper investigation of the output waveform, we observe that

[3]The ⟨start⟩ token is avector of all ones while the ⟨end⟩token is a vector of all zeroes. We also perform teacher enforcing for better results.

[4]We ensured that the setup allowed them to play tabla without any hindrance.

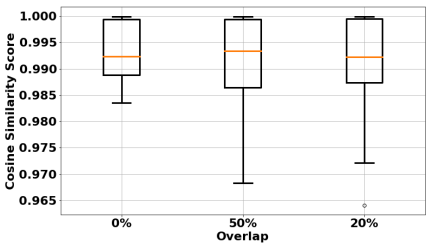
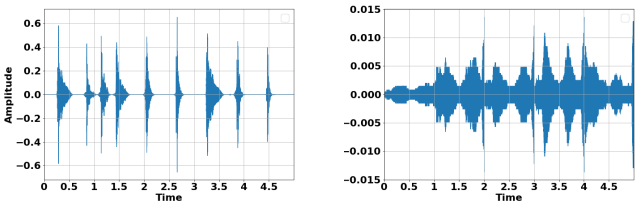


Fig. 6: Cosine similarity between the MFCC of the ground-truth audio and the audio generated using *vBeats*.



(a) (b)

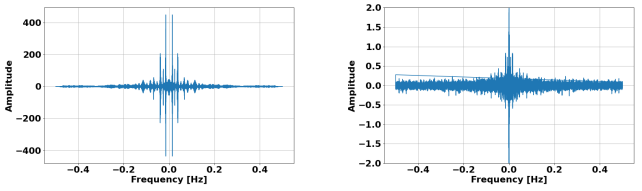
Fig. 7: Performance evaluation: Similarity in time-domain (a) Ground-truth waveform from the recorded audio and (b) Generated waveform with *vBeats*.

both in the temporal (See Fig. 7) as well as in the spectral domain (See Fig. 8) that the generated acoustic signature has significant similarity with the ground-truth signature. However, we can also observe that the generated signatures additionally contain many noise components that can significantly impact the overall quality of the generated audio.

4. Framework Insights Next, we analyze the performance of the ARM in Fig. 9. The results show that the designed autoencoder performs reasonably well and can reconstruct the acoustic signatures with a some amount of noise.

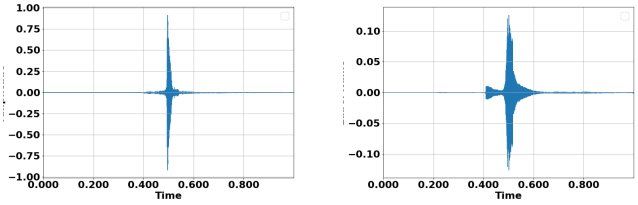
VII. Limitations and Future Work

Although *vBeats* can achieve a significant similarity in terms of the similarity of waveforms, following are some improvements that can be added to the framework which need additional research and system deployment and therefore, is included as a part of the future work.



(a) (b)

Fig. 8: Performance evaluation: Similarity in frequency-domain (a) Ground-truth waveform from the recorded audio and (b) Generated waveform with *vBeats*.



(a) (b)

Fig. 9: Time-domain analysis of noise introduced in ARM (a) Original audio and (b) Recreated audio.

1. Reducing the Impact of Noise One major improvement that the current version of *vBeats* requires is in the reduction of noise from the re-created audio. Although a major reason for the lack of clarity can be attributed to the limited amount of training data available for developing the seq2seq model, a significant cause can also be found in the relatively straightforward design of the acoustic autoencoder which ultimately restricts the quality of the latent embedding and the temporal correlation of the strokes.

2. Local Frame of Reference Unlike drums, tabla does not have an uniform plane in the membrane as each head has a central area that is inked (called "*syahi*") which allows it to produce harmonic overtones. Naturally, it is important to understand where the fingers, palm or the wrist strikes. However, achieving this can be challenging as we may need to create a virtual surface for the reference coordinates of the gestures to be mapped.

3. Two-Hands in Action Notably, playing tabla also needs both hands working in perfect synchronization. With tabla, the percussionist usually strikes the pair of drums (a.k.a "*baya*" and "*daya*") and the final tone is the convolution of the sounds generated by both these drums. This can be adapted using the existing *vBeats* architecture for both the hands separately, however that may over simplify the general complex nature of acoustic symphony generated while a tabla is being played.

VIII. Conclusion

In this paper, we present *vBeats*, a framework that allows a user to simulate the bass strokes of a tabla from the in-air gestures using a COTS smartphone. In the backend, *vBeats* intelligently uses auto-encoders trained using publicly available datasets to obtain meaningful embeddings and perform cross-modal translation from gesture to audio using a typical seq2seq model. A principled evaluation, using an in-house collected dataset, of the initial version of *vBeats* shows the potential of the overall idea. Also, it highlights the challenges of such a framework that may need fine-tuning in future versions.

References

[1] E. Lewis, "Independent music-making during covid-19 and mental health," *Psychology of Music*, p. 03057356221126198, 2022.

[2] "Play Guitar!" https://apps.microsoft.com/store/detail/play-guitar/9WZDNCRFJ9ZD?hl=en-us&gl=us, 2012, online; Last Accessed: December 7, 2023.

[3] S. Serafin, S. Trento, F. Grani, H. Perner-Wilson, S. Madgwick, and T. J. Mitchell, "Controlling physically based virtual musical instruments using the gloves," 2014.
[4] D. Brown, C. Nash, and T. Mitchell, "Understanding user-defined mapping design in mid-air musical performance," in *ACM MOCO*, 2018, pp. 1–8.
[5] I. Heap, "MiMu Gloves," https://mimugloves.com/gloves/, 2012, online; Last Accessed: December 7, 2023.
[6] S. Bhalla, M. Goel, and R. Khurana, "Imu2doppler: Cross-modal domain adaptation for doppler-based activity recognition using imu data," *PACM IMWUT*, pp. 1–20, 2021.
[7] J. Gillick, A. Roberts, J. Engel, D. Eck, and D. Bamman, "Learning to groove with inverse sequence transformations," in *ICML*, 2019.
[8] R. Hennequin, A. Khlif, F. Voituret, and M. Moussallam, "Spleeter: a fast and efficient music source separation tool with pre-trained models," *J. Open Source Softw.*, p. 2154, 2020.
[9] T. Sainburg, M. Thielk, and T. Q. Gentner, "Finding, visualizing, and quantifying latent structure across diverse animal vocal repertoires," *PLoS computational biology*, 2020.
[10] C. Recommendation, "Pulse code modulation (pcm) of voice frequencies," in *ITU*, 1988.
[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016, pp. 770–778.