IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 13, No. 3, September 2024, pp. 3371∼3380
ISSN: 2252-8938, DOI: 10.11591/ijai.v13.i3.pp3371-3380

Acapella-based music generation with sequential models utilizing discrete cosine transform

Julian Saputra, Agi Prasetiadi, Iqsyahiro Kresna
Departement of Computer Science, Faculty of Informatics, Institut Teknologi Telkom Purwokerto, Purwokerto, Indonesia

Article Info

Article history:

Received Jun 6, 2023
Revised Dec 3, 2023
Accepted Feb 10, 2024

Keywords:

Autoencoder
Discrete cosine transform
Gated recurrent unit
Music instrument
Recurrent neural network

ABSTRACT

Making musical instruments that accompany vocals in a song depends on the mood quality and the music composer's creativity. The model created by other researchers has restrictions that include being limited to musical instrument digital interface files and relying on recurrent neural networks (RNN) or transformers for the recursive generation of musical notes. This research offers the world's first model capable of automatically generating musical instruments accompanying human vocal sounds. The model we created is divided into three types of sound input: short input, combed input, and frequency sound based on the discrete cosine transform (DCT). By combining the sequential models such as autoencoder and gated recurrent unit (GRU) models, we will evaluate the performance of the resulting model in terms of loss and creativity. The best model has a performance evaluation that resulted in an average loss of 0.02993620155. The hearing test results from the sound output produced in the frequency range 0-1,600 Hertz can be heard clearly, and the tones are quite harmonious. The model has the potential to be further developed in future research in the field of sound processing.

Corresponding Author:

Julian Saputra
Departement of Computer Science, Faculty of Informatics, Institut Teknologi Telkom Purwokerto
Purwokerto, Central Java 53147, Indonesia
Email: 19102008@ittelkom-pwt.ac.id

1. INTRODUCTION

Creating background music involves a series of complex stages in music composition, requiring expertise in music theory and the ability to write musical notes [1]–[3]. Music itself encompasses various elements like melody, rhythm, harmony, tempo, and timbre, which are essential for creating harmony between instruments and lyrics in a song [4]–[6]. Notably, a cappella music relies solely on human voices, no instruments, producing a unique experience [7], [8].

Research in machine learning and deep learning has sparked interest in music studies [9]–[11]. Deep learning is being employed for automatic music generation using convolutional neural networks (CNN) for feature extraction and recurrent neural networks (RNN) for symbolic melodies [12], [13]. CNN algorithms are adept at voice-to-text applications [14], while RNNs handle time series data, retaining memory of input sequences [15], [16]. Surprisingly, there's a research gap in translating human voices into background music.

This research explores the creation of automatic instrumental music generators from a cappella using deep learning, involving Conv1D for audio signal processing and RNN for sequential sound data [17], [18]. Three input schemes are studied: short input, combed input, and discrete cosine transform (DCT) input. Each scheme will be explored with various model evaluations to identify the most effective techniques for high-quality instrumental music generation from human voice recordings.

2. METHOD

2.1. Data collection

In this study, the main dataset comprises human vocal and acoustic guitar instrument sounds from song covers. The music data is read from two channels, each with a 44,100 Hz sampling frequency, resulting in 88,200 data points per second. The data is then normalized to a range of -1 to 1 to prepare it for deep learning processing [19]. The dataset includes 250 human vocals and 250 guitar instrument sounds sourced from YouTube in MP3 format. These sounds are separated using Spleeter, a Python library developed by Deezer R&D, which utilizes TensorFlow-based models to extract vocals and instruments [20].

2.2. Autoencodoer

An autoencoder, a neural network model with input and output layers, aims to learn and reconstruct input data. Typically, autoencoders are used for feature dimension reduction, involving two main components: the encoder and decoder models [21]. The encoder maps input data X to a smaller latent representation Z. This reduction captures essential data features while decreasing dimensionality. The decoder, on the other hand, reconstructs Z back to the original input dimension, yielding X'. The encoder functions as a compression tool, while the decoder serves as an extraction tool [21], [22].

In more complex autoencoder forms, the encoder can have multiple hidden layers, transforming input data $X \in \mathbb{R}^{N\times P}$, where N is the number of samples, into a latent data representation $Z \in \mathbb{R}^{N\times P}$. The latent space is defined as $Z = \sigma(WX + b)$, with K being the original feature dimension and P representing the compressed feature dimension. Here, W is a weight matrix, b is a bias vector, and σ represents an activation function, like rectified linear unit (ReLU). The decoder is to map latent representation Z back to the original input space with $X' = \sigma(W'Z + b')$. Here, $X' \in \mathbb{R}^{N\times P}$ represents the decoder's output or approximation of the original data, minimizing the difference between X and X' denoted by $L = \| X - X' \|^2$. This process optimizes parameters $\theta = \{W, b, W', b'\}$ during training.

2.3. Recurrent neural network

RNN is a deep learning architecture ideal for sequential data processing [23], [24]. It is used for tasks like language translation, speech recognition, and stock market prediction, as it remembers information from previous inputs [25]. In an RNN model, each data point in a sequence goes through a series of neural networks. For each element in the sequence, a single neural network layer processes the current input $x^{(i)}$ and the previous hidden state $h^{(i-1)}$ to produce output $y^{(i)}$. The formulas for this process are: $h^{(i)} = \sigma_h(W_{hh} \cdot h^{(i-1)} + W_{hx}x^{(i)} + b_h)$ and $y^{(i)} = \sigma_y(W_{yh}h^{(i)} + b_h)$. Here, W_{hh}, W_{hx}, and W_{yh} perform linear transformations, while b_h and b_y are bias terms. Activation functions σ_h and σ_y introduce non-linearity, and $h^{(0)}$ is initialized as a vector of 0. However, RNNs can face vanishing gradient problems in deep learning [22], [26]. To tackle this, two RNN cell types have been developed: long short-term memory (LSTM) and gated recurrent unit (GRU).

GRU is a simplified version of LSTM, performing similarly better in some cases. GRU employs two gate units: the update gate and the reset gate, which control information flow [22]. The update gate regulates what is carried over from the previous time step to the next. It's defined as $z_t = \sigma(W_z \cdot x^t + U_z \cdot h^{(t-1)} + b_z)$. The reset gate determines what information from the past should be kept, and its formula is $r_t = \sigma(W_r \cdot x^t + U_r \cdot h^{(t-1)} + b_r)$. Afterward, the reset gate helps store valuable information in the new memory content: $\bar{h}^{(t)} = \tanh(W \cdot x^t + U \cdot (r_t \odot h^{(t-1)} + b)$, where $\odot$ denotes a bitwise matrix multiplication [26]. Finally, the network calculates the output, represented by $h^{(t)}$, containing current time step information and data from past steps, determined by the gates: $h^{(t)} = (1 - z_t) \odot \bar{h}^{(t)} + z_t \odot h^{(t-1)}$ [27].

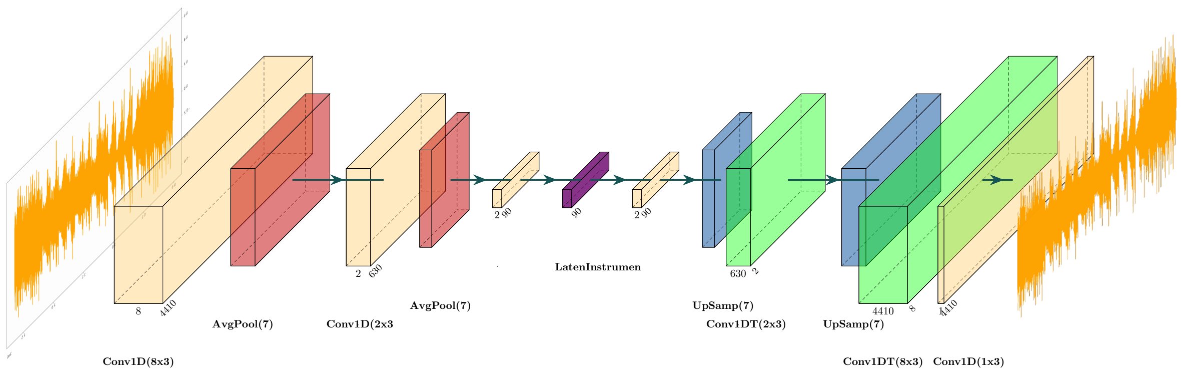
2.4. Scheme design

The method used to achieve the aim of this research is by designing a model that takes vocal sound as input and generates harmonious musical instruments as output. It employs the mean absolute error (MAE) loss function for training. There are three techniques for processing input and output data. The first is autoencoder-RNN with data divided into 4,410 or 8,820; this is termed short input. Two autoencoders are created for vocal sound and background music. The second technique is autoencoder-combing-RNN, where data is cut at 64*4,410, called combed input. Two autoencoders are also created for vocal sound and background music.

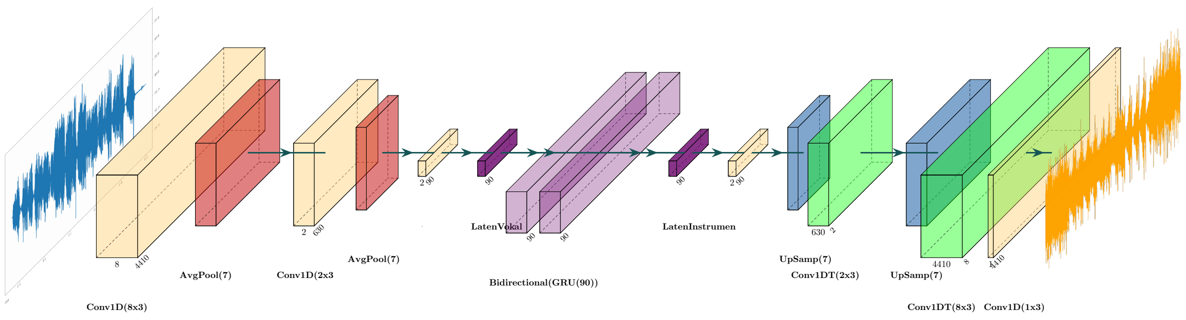
The third technique is DCT-RNN, dividing data into 30 or 60-second intervals, using DCT format and filtering frequencies from 0 to 800 or 0 to 1,600; this is referred to as DCT input. The use of autoencoders with short inputs is due to memory constraints during training.

2.4.1. Short input

The initial step involves creating two autoencoders, with the training process of one of the autoencoders depicted in Figure 1(a), to simplify input music data into latent data. One autoencoder processes vocal sound, and the other processes instrument data, both aiming for minimal-sized latent data. These compact representations enable quicker RNN-GRU model training with reduced memory consumption. The encoder models from both autoencoders are then employed to train latent data features from vocal and instrument sounds, targeting minimal loss. All autoencoders merge into a primary model with short input. The first autoencoder's encoder connects to a GRU model, which, in turn, links to the second autoencoder's decoder, facilitating feature extraction using the GRU model and output generation with the second autoencoder's decoder, as shown in Figure 1(b).



(a)

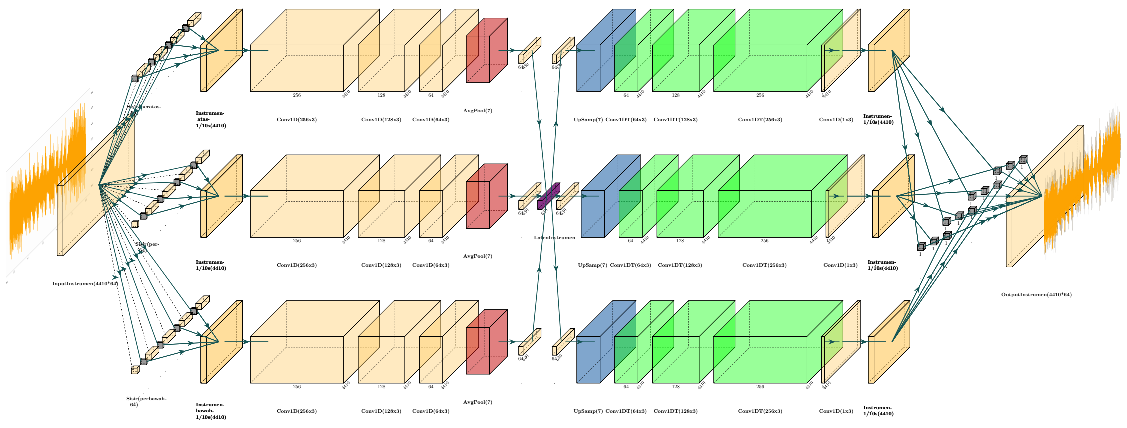


(b)

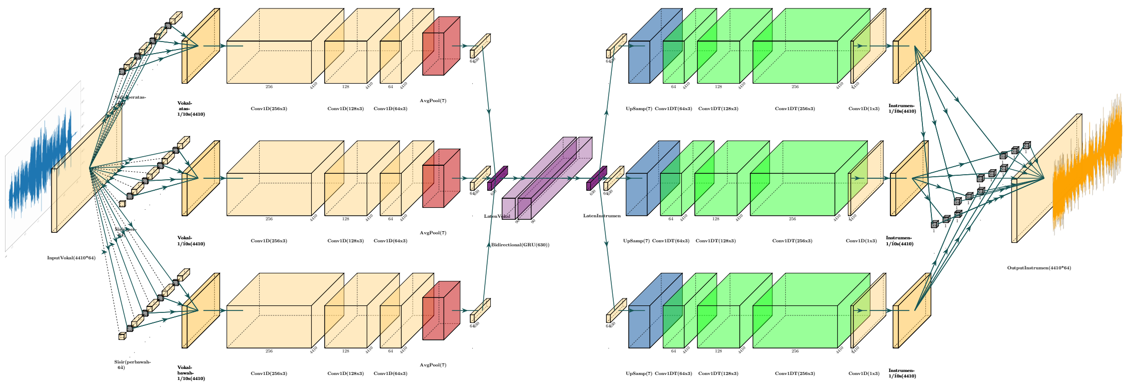
Figure 1. Training and application of autoencoder and RNN models (a) training autoencoder with vocal and instrument sound and (b) generating instrument based on vocal sound using short input

2.4.2. Combed input

This scheme alters the input model configuration, using a combed input with sequential arrangements. The first two steps are depicted in Figure 2(a), where input is divided into parts resembling combing. The illustration displays only the initial three sections of the combed area on a single autoencoder, yet during training, all 64 segments covering 6.4 seconds are utilized on two autoencoders. The ultimate aim remains to obtain compact latent data. In the next step, each autoencoder's encoder is employed to train latent features for vocals and instruments. Finally, all trained models are combined into the primary model with combed input, as shown in Figure 2(b).



(a)



(b)

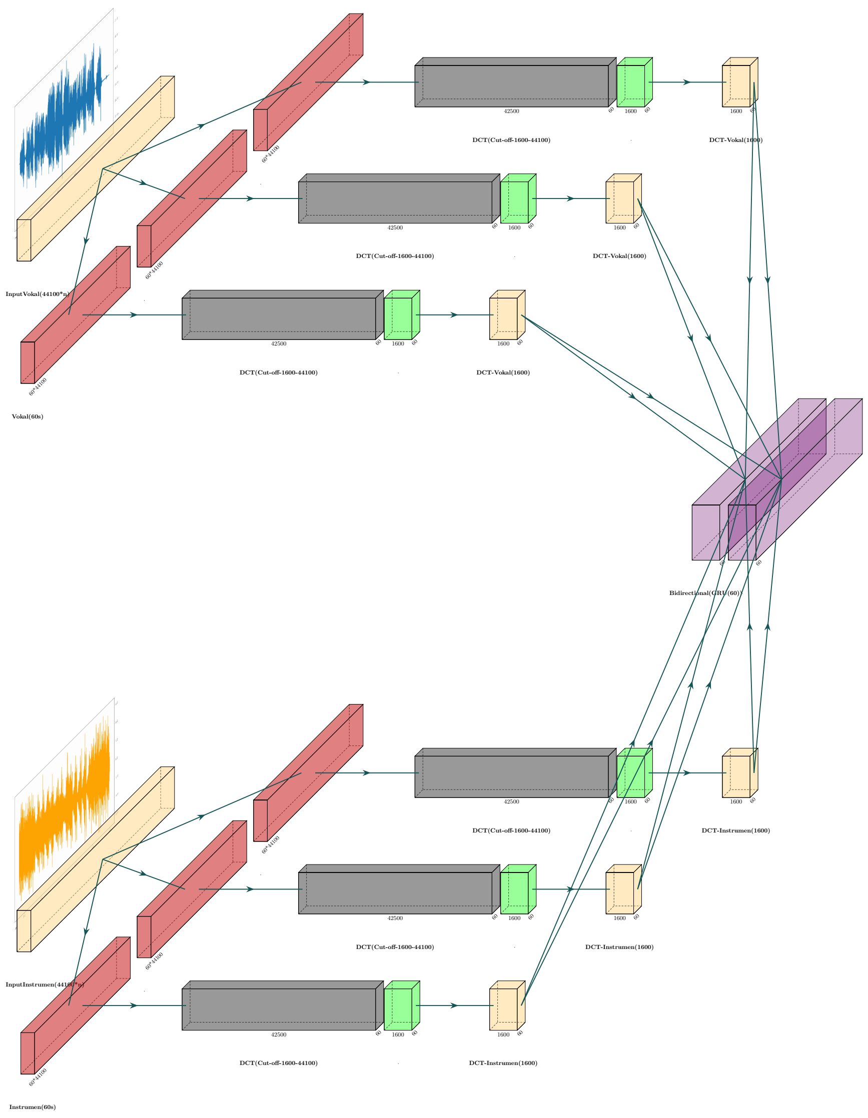
Figure 2. Training and application of autoencoder and RNN models (a) training autoencoder with combed vocal and instrument sound and (b) generating instrument based on vocal sound using combed input

2.4.3. Discrete cosine transform input

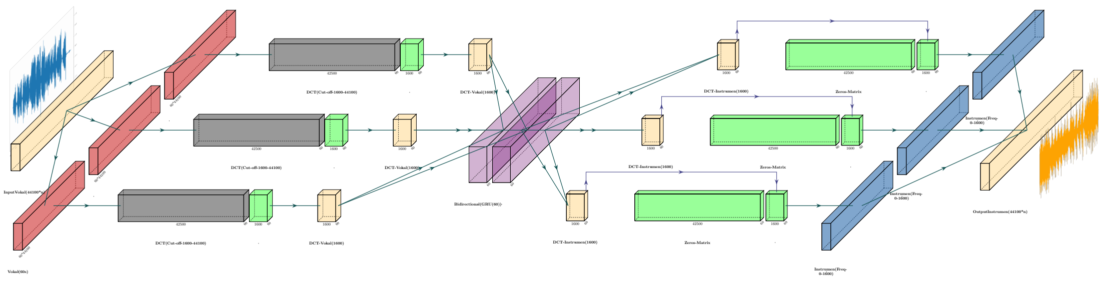
In this phase, the autoencoder technique is no longer used to extract song data. Instead, song data segments of durations lasting 30 or 60 seconds will be directly utilized. Each data segment will be transformed into DCT form per second, and only the frequency range of 0-800 Hertz or 0-1,600 Hertz will be taken. The prepared data segments will have dimensions of 30×800 or $60 \times 1,600$. An RNN with fewer layers will process this smaller data size faster than data with dimensions of 800×30 or $1,600 \times 60$. The first step of this process is illustrated in Figure 3(a). After training the RNN model, the finalization of the primary model in this technique can be directly performed with the architecture structure as shown in Figure 3(b). This model's final output is only the musical instrument's sound within the frequency range of 0-800 Hertz or 0-1,600 Hertz. Then, for more complex and varied output results, another model is explicitly created to receive input music data with frequencies of 1,600-3,200 Hertz, 3,200-4,800 Hertz, and 4,800-6,400 Hertz. Using this approach, several different output models will be combined into one, resulting in a complete sound of the instrument with all frequencies included.

3. RESULTS AND DISCUSSION

Creating an effective base model is crucial before experimenting with DCT techniques. The base model will be further enhanced, but first, we must define what qualifies as a good output. In short input methods, creativity in output is limited due to the short 0.1 or 0.2-second response time. Handling sound data for a brief duration consumes significant memory. An acceptable result can be characterized as low-noise instrumental music. It's essential to avoid overly small latents in autoencoders, as this may result in unclear or empty output.



(a)



(b)

Figure 3. Training and application of RNN models (a) the GRU model is trained with DCT vocal and instrument latents and (b) the final model where the instrument music sound is generated based on vocal sound using DCT input

3.1. Base model

Table 1 displays results for the basic model trained with 50 and 250 songs, each with 0.1-second slices. AE1-RNN1 and AE2-RNN2 are two models, with ”AE-RNN” representing autoencoder and RNN. The analysis reveals higher loss in vocal sounds than in instrument sounds, indicating better reconstruction of instruments due to the complexity of vocals. While AE2-RNN2 has a lower average loss overall, AE1-RNN1, with more limited dataset, produces crisper and better sound in live tests. This implies the first model's advantage with more limited dataset. Increasing data size might hinder pattern capture, especially in creative music, despite ample computational capacity. The current architecture yields a 90 x 2 latent vector, a substantial compression from the original 4,410 data, and a 24.5x compression ratio. The model scale will change in the next experiment to suit an 8,820-sized input.

Table 1. Performance evaluation of base model

Model	Encoder Architecture	RNN Architecture	AE average						RNN	
			Loss	Vocal loss	Instrumental loss	Bias	Vocal bias	Instrumental bias	Loss	Bias
AE1-RNN1	C8A7C2A7	GRU90	0.07264	0.05257	0.02006	0.02673	0.02030	0.00795	0.05344	0.01934
AE2-RNN2	C16A7C32A7	GRU90	0.00825	0.00718	0.00107	0.00434	0.00388	0.00065	0.00028	0.00012

3.2. Base model expansion

We proceeded to further investigate architectural configurations with a more considerable 8,820-sized input due to the promising outcomes of the prior. We conducted an experiment using ten datasets, and the model's performance is summarized in Table 2. Initially, AE3-RNN3 and AE4-RNN4 models were tested, compressing input by about 49 times. Although it had a higher losses than AE2-RNN2, both produced better audio fidelity. Later, input compression was increased to 70 times for AE5-RNN5 and reduced to 35 times for AE6-RNN6. Surprisingly, AE6-RNN6, with higher RNN loss, outperformed AE5-RNN5. The top-performing model, AE6-RNN6, was further trained using 50 song samples, known as the AE7-RNN7 model. Although its autoencoder loss is relatively high, it is lower than the initial AE1-RNN1 model. However, RNN loss and bias are higher compared to the other six models. Interestingly, this model produces the clearest results with minor noise, following AE1-RNN1. The performance of model AE7-RNN7 can be seen in Figure 4.

Table 2. Performance evaluation of base model expansion from AE3-RNN3 and AE7-RNN7 models

Model	Encoder Architecture	RNN Architecture	AE average						RNN	
			Loss	Vocal loss	Instrumental loss	Bias	Vocal bias	Instrumental bias	Loss	Bias
AE3-RNN3	C8A7C32A7C64	GRU180D64	0.04609	0.00718	0.00107	0.01408	0.01174	0.003	0.0091	0.00177
AE4-RNN4	C16A7C8A7C32	GRU180D32	0.04094	0.03147	0.00946	0.01231	0.01029	0.00259	0.01099	0.00258
AE5-RNN5	C64A7C8A5C32A2	GRU126D32	0.04094	0.03147	0.00946	0.01231	0.01029	0.00259	0.02218	0.00363
AE6-RNN6	C128C4A7C8A5 (10 dataset)	GRU252D8	0.0378	0.0298	0.0079	0.0116	0.0097	0.0022	0.0429	0.00918
AE7-RNN7	C128C4A7C8A5 (50 dataset)	GRU252D8	0.04363	0.03209	0.01153	0.01707	0.01299	0.00473	0.29213	0.05626

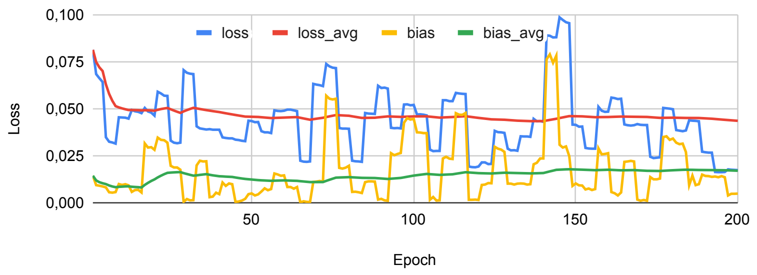
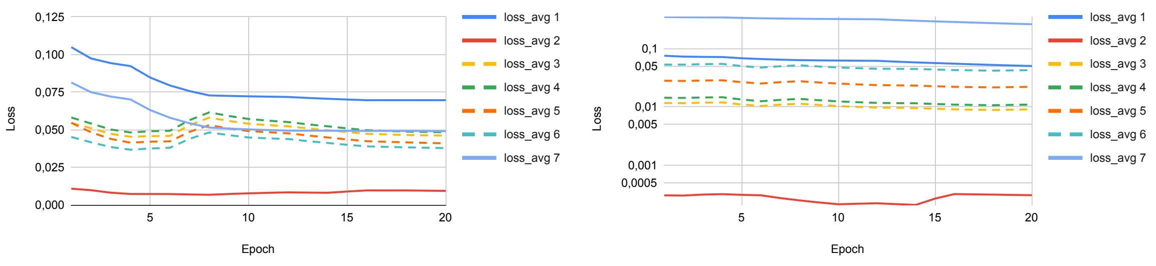


Figure 4. Autoencoder AE7-RNN7's performance in detail with short input

Figure 5(a) displays autoencoder model performance using the cumulative moving average (CMA). The CMA of the loss is defined as $c_j = \sum_{i=0}^{j} x_i/(j+1)$ where c_j represents the CMA value at index j and x_i corresponds to the loss value at index i. AE1-RNN1 and AE7-RNN7 architectures demonstrate acceptable loss performance, suggesting effective pattern learning. However, other models show inconsistent loss curves, suggesting difficulties in capturing background music patterns. Therefore, models resembling AE1-RNN1 or

AE7-RNN7 might yield promising results. In Figure 5(b), AE1-RNN1 and AE7-RNN7 display smaller RNN loss than other models, signifying pattern learning ability. Nevertheless, their RNN loss remains higher. Higher loss values may imply greater imaginativeness in composing instrumental music, while lower values suggest limited exploration of creativity.



(a) (b)

Figure 5. CMA of losses across AE1-RNN1 to AE7-RNN7 architectures where (a) autoencoder and (b) RNN

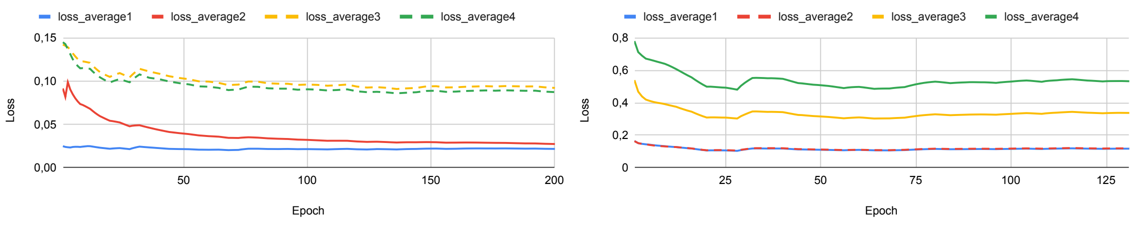
3.3. Combed input

This investigation used four different combinations of autoencoder and RNN architecture. Two models featured 4,410-sized inputs, while the other two had 8,820. Despite these differences, all followed the same architecture to determine the best one for generating background music. In Table 3, there are four combed (C) models, named C1, C2, C3, and C4. Each model combines convolution 1-dimension and average pooling on the encoder side and GRU and dense layers on the RNN side. C1 had the lowest average loss and lower bias averages than other models, suggesting that increasing RNN architecture's dense layers can improve performance. In contrast, C4, with fewer dense layers, had the highest bias and loss values.

Table 3. The model structure employed in the combed input approach

Model	Architecture (Encoder Side)	RNN	Input	Size
C1	Conv1D256Conv1D128Conv1D64A7	GRU630D64	4,410	100
C2	Conv1D256Conv1D128Conv1D64A7	GRU630D64	4,410	200
C3	Conv1D256Conv1D96Conv1D64A7A2	GRU630D32	8,820	100
C4	Conv1D256Conv1D96Conv1D64A7A2	GRU630D32	8,820	200

Figure 6 reveals that Model C1 outperforms other models, exhibiting the lowest loss. C2 exhibited a more gradual convergence yet eventually achieved a similar loss as C1, as shown in Figure 6(a). However, C3 and C4 performed less effectively. In terms of RNN performance, C1 had the lowest loss, closely followed by C2, as can be seen in Figure 6(b). In listening tests, C1 produced more distinct results with reduced noise and lower frequency than the others, whereas C3 and C4 generated background music with more noise and higher frequency.



(a) (b)

Figure 6. CMA of losses across C1 to C4 architectures, (a) autoencoder and (b) RNN

3.4. Discrete cosine transform input

In this experiment, there were 32 models of DCT with different architectures and parameters, which can be seen in Table 4. The dense layer should have the exact number of nodes as the remaining frequencies after the cut-off. In this case, the cut-off data includes samples with sizes of 800 (from frequency 0-800) and 1,600 (from frequency 0-1,600). The window (w) to divide the data on the voice signal into several parts with sizes 30 and 60, the GRU architecture with the number of networks is 30 and 60. In this research, the datasets used are divided into two parts: 50 datasets for training and 129 datasets consisting of the results of grouping the data based on their further similarity using the K-means clustering technique.

Table 4. The model architecture used in DCT input scheme with performance from DCT1 to DCT32

Model	RNN	Window	Data Train	Epoch	Model	RNN	Window	Data Train	Epoch
DCT1	GRU30D800	30	50	60	DCT17	GRU60D1600	60	129	120
DCT2	GRU30D1600	30	50	60	DCT18	GRU60D1600	60	50	120
DCT3	GRU60D800	60	50	60	DCT19	GRU30D1600	60	129	120
DCT4	GRU60D1600	60	50	60	DCT20	GRU60D1600	30	129	120
DCT5	GRU60D800	60	129	60	DCT21	GRU60D800	60	129	120
DCT6	GRU60D1600	60	129	60	DCT22	GRU30D1600	60	50	120
DCT7	GRU60D800	60	129	60	DCT23	GRU60D1600	30	50	120
DCT8	GRU60D1600	60	129	60	DCT24	GRU60D800	60	50	120
DCT9	GRU30D800	60	50	60	DCT25	GRU30D1600	30	129	120
DCT10	GRU60D800	30	50	60	DCT26	GRU30D800	60	129	120
DCT11	GRU30D800	60	129	60	DCT27	GRU60D800	30	129	120
DCT12	GRU60D800	30	129	60	DCT28	GRU30D1600	30	50	120
DCT13	GRU30D1600	30	129	60	DCT29	GRU30D800	60	50	120
DCT14	GRU30D1600	60	50	60	DCT30	GRU30D800	30	50	120
DCT15	GRU30D1600	60	129	60	DCT31	GRU60D800	30	50	120
DCT16	GRU60D1600	30	50	60	DCT32	GRU30D800	30	129	120

Examining the experimental results with CMA loss in Figure 7, the models display underdamped behavior, suggesting that they are slow to stabilize after experiencing oscillations. This behavior indicates that the models initially reduce their loss significantly and gradually stabilize as they attempt to adapt to the diverse song dataset. However, this underdamped response may imply potential weaknesses in capturing critical sound signals, reducing the model's overall performance quality. In Figure 8, the model performance results are displayed, including average loss, average bias, and standard deviation. The DCT17 model demonstrates consistent learning with minimal deviation between loss and validation loss, signifying its strong performance in data processing. In the final stage of this scheme, the DCT17 model's specifications are used to retrain it for the frequency ranges of 1,600-3,200 Hertz, 3,200-4,800 Hertz, and 4,800-6,400 Hertz. When testing these four models, the 0-1,600 Hertz models produce harmonious, pitched sounds, while the others have a higher frequency feel. As the frequency range increases, the sound remains similar. These four models are then combined to create more complex and diverse sounds. The results produced by all models can be listened on this link https://s.id/sc_autobg.

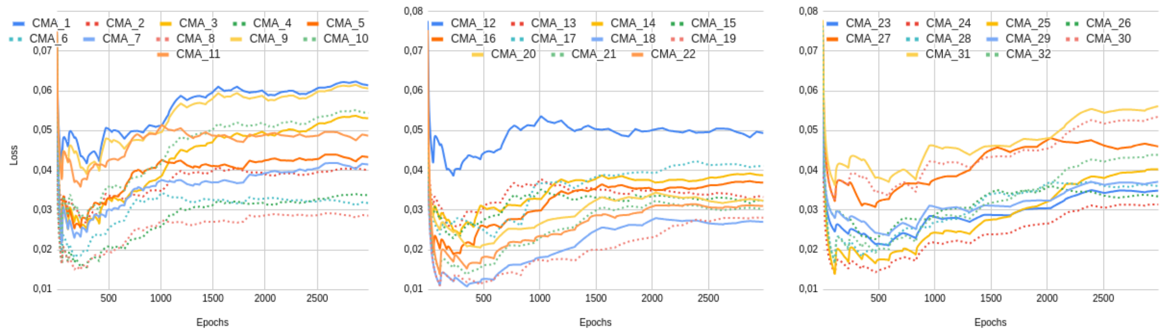


Figure 7. CMA of loss from model DCT1 to DCT32

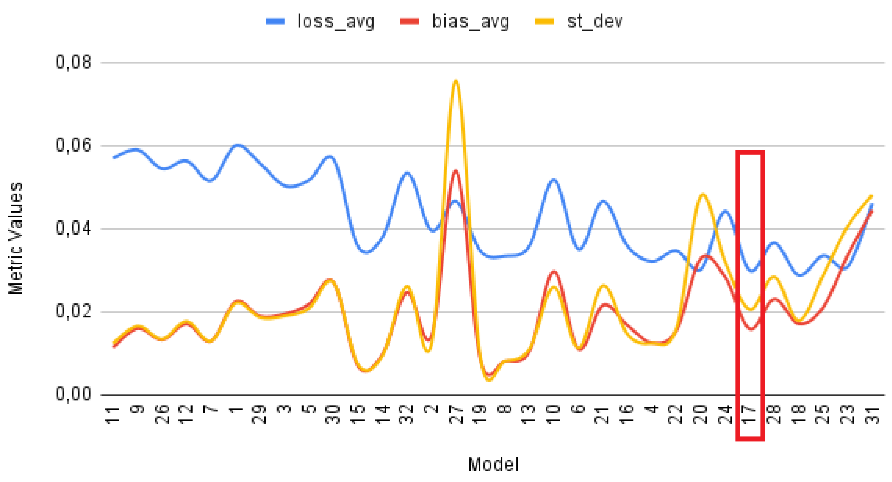


Figure 8. DCT model performance based on average loss, average bias, and standard deviation: ranking by standard deviation

4. CONCLUSION

Autoencoder techniques are employed in this research to compress and decompress data for the development of an automatic musical instrument based on human vocal sounds. The findings reveal that an improved encoder architecture consists of cleaner convolutional layers in decreasing order, such as C(N)C(N/2)C(N/4), and so on. This aspect is crucial as the RNN architecture relies on latent data of size n x m, where a larger value of n leads to longer training times. However, a larger value of m yields better results. Therefore, for future investigations, researchers are encouraged to explore the use of a significantly smaller n compared to m, aligning it with the time step in the RNN layer. Furthermore, enhancing musical creativity can be achieved by increasing the number of nodes within the RNN network.

REFERENCES

[1] E. Threadgold, J. E. Marsh, N. Mclatchie, and L. J. Ball, "Background music stints creativity: evidence from compound remote associate tasks," *Applied Cognitive Psychology* vol. 33, pp. 873-888. 2019, doi: 10.1002/acp.3532.

[2] C. Gupta, E. Yılmaz, and H. Li, "Automatic lyrics alignment and transcription in polyphonic music: does background music help?," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 496-500, 2020, doi: 10.1109/ICASSP40776.2020.9054567.

[3] Y. Tang, B. M. Fazenda, and T. J. Cox, "Applied sciences automatic speech-to-background ratio selection to maintain speech intelligibility in broadcasts using an objective intelligibility metric," *Applied Sciences*, vol. 8, no. 1, 2018, doi:10.3390/app8010059.

[4] S. Dai and G. G. Xia, "Music style transfer: a position paper," *arXiv preprint*, 2018, doi: arxiv.org/abs/1803.06841.

[5] T. Särkämö, "Music for the ageing brain: cognitive, emotional, social, and neural benefits of musical leisure activities in stroke and dementia," *Dementia*, vol. 17, no. 6, pp. 670-685, 2018, doi: 10.1177/1471301217729237.

[6] K. Lems, "New ideas for teaching English using songs and music," *English teaching forum*, vol. 56, no. 1, pp. 14–21, 2018.

[7] J. Pons, R. Gong, and X. Serra, "Score-informed syllable segmentation for a cappella singing voice with convolutional neural networks," in *Proceedings of the 18th ISMIR, Suzhou, China*, pp. 383–389, 2017.

[8] J. F. Montesinos, V. S. Kadandale, and G. Haro, "A cappella: audio-visual singing voice separation," *arXiv preprint*, 2021, doi: 10.48550/arXiv.2104.09946.

[9] T. Stegemann, M. Geretsegger, E. P. Quoc, H. Riedl, and M. Smetana, "Music therapy and other music-based interventions in pediatric health care: an overview," *Medicines*, vol. 6, no. 1, 2019, doi: 10.3390/medicines6010025.

[10] L. Bresler, "Qualitative paradigms in music education research," *Visions of Research in Music Education*, vol. 16, no. 1, 2021.

[11] A. Prasetiadi, J. Saputra, I. Kresna, and I. Ramadhanti, "High dimensional vectors based on discrete cosine transform in generating background music from vocal sound," in *2023 International Electronics Symposium (IES)*, pp. 434-439, Aug. 2023.

[12] B. R. Kilambi, A. R. Parankusham, and S. K. Tadepalli, "Instrument recognition in polyphonic music using convolutional recurrent neural networks," in *Proceedings of International Conference on Intelligent Computing, Information and Control Systems, Springer, Singapore*, vol. 1272, pp. 375-383, 2021, doi: 10.1007/978-981-15-8443-5_38.

[13] J. Wu, C. Hu, Y. Wang, X. Hu and J. Zhu, "A hierarchical recurrent neural network for symbolic melody generation," in *IEEE Transactions on Cybernetics*, vol. 50, no. 6, pp. 2749-2757, June 2020, doi: 10.1109/TCYB.2019.2953194.

[14] S. Bunrit, T. Inkian, N. Kerdprasop, and K. Kerdprasop, "Text-independent speaker identification using deep learning model of convolution neural network," *International Journal of Machine Learning and Computing*, vol. 9, no. 2, pp. 143-148, 2019, doi: 10.18178/ijmlc.2019.9.2.778.

[15] Z. Lv, J. Xu, K. Zheng, H. Yin, P. Zhao, and X. Zhou, "LC-RNN: a deep learning model for traffic speed prediction," *IJCAI*, pp. 3470-3476, 2018, doi: 10.24963/ijcai.2018/482.
[16] G. Kurata and G. Saon, "Knowledge distillation from offline to streaming RNN transducer for end-to-end speech recognition," *Interspeech*, pp. 2117-2121, 2020, doi: 10.21437/Interspeech.2020-2442.
[17] L. Deng and D. Yu, "Deep learning: methods and applications," *Foundations and trends® in signal processing*, vol. 7, no. 3-4, pp. 197–387, 2014, doi: 10.1561/2000000039.
[18] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets*, vol. 31, no. 3, pp. 685-695, 2021, doi: 10.1007/s12525-021-00475-2.
[19] S. Picard, C. Chapdelaine, C. Cappi, L. Gardes, E. Jenn, B. Lefevre, T. Soumarmon, "Ensuring dataset quality for machine learning certification," in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 275-282, 2020, doi: 10.1109/ISSREW51248.2020.00085.
[20] R. Hennequin, A. Khlif, F. Voituret, and M. Moussallam, "Spleeter: a fast and efficient music source separation tool with pre-trained models," *Journal of Open Source Software*, vol. 5, no. 50, pp. 2154, 2020, doi: 10.21105/joss.02154.
[21] Y. Su, J. Li, A. Plaza, A. Marinoni, P. Gamba, and S. Chakravortty, "DAEN: deep autoencoder networks for hyperspectral unmixing," in *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 7, pp. 4309-4321, 2019, doi: 10.1109/TGRS.2018.2890633.
[22] D. Molho *et al.*, "Deep learning in single-cell analysis," *arXiv preprint*, 2022, doi: 10.48550/arXiv.2210.12385.
[23] T. Donkers, B. Loepp, and J. Ziegler, "Sequential user-based recurrent neural network recommendations," in *Proceedings of the eleventh ACM conference on recommender systems*, pp. 152-160, 2017, doi: 10.1145/3109859.3109877.
[24] M. Kaur and A. Mohta, "A review of deep learning with recurrent neural network," in *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp. 460-465. 2019, doi: 10.1109/ICSSIT46314.2019.8987837.
[25] F. Ilhan, O. Karaahmetoglu, I. Balaban, and S. S. Kozat, "Markovian RNN: an adaptive time series prediction network with HMM-based switching for nonstationary environments," in *IEEE Transactions on Neural Networks and Learning Systems.* 2021, doi: 10.1109/TNNLS.2021.3100528.
[26] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, "Independently recurrent neural network (IndRNN): building a longer and deeper RNN," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5457-5466, 2018, doi: 10.1109/CVPR.2018.00572.
[27] P. T. Yamak, L. Yujian, and P. K. Gadosey, "A comparison between ARIMA, LSTM, and GRU for time series forecasting," in *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, pp. 49-55, 2019, doi: 10.1145/3377713.3377722.

BIOGRAPHIES OF AUTHORS

Julian Saputra comes from Sampit, currently studying for a Bachelor's Degree in Informatics Engineering at the Institut Teknologi Telkom Purwokerto. Active activities on campus that he has participated in include lecturer assistant and practicum assistant for image and video data analysis course. He joined the campus Institutional organization, namely the Student Council and joined as an Independent study at the artificial intelligence center Indonesia. His research areas include data science engineer, machine learning, deep learning, natural language processing and education. He can be contacted at email: 19102008@ittelkom-pwt.ac.id.

Agi Prasetiadi holds a Bachelor of Electronics degree in General Electrical Engineering, VLSI and Computer Vision at Bandung Institute of Technology. He holds a Master of Engineering (M.Eng.) in Realtime Systems, Bio-Inspired Technology at Kumoh National Institute of Technology. He currently teaches at the Institut Teknologi Telkom Purwokerto as a lecturer in Informatics. His research areas include computer vision, machine learning, and deep learning. He can be contacted at email: agi@ittelkom-pwt.ac.id.

Iqsyahiro Kresna holds a Bachelor of Informatics Engineering degree in Informatics Faculty at Telkom University. He holds a Master of Engineering (M.T.) in Informatics Engineering at Bandung Institute of Technology. He currently teaches at the Institut Teknologi Telkom Purwokerto as a lecturer in Informatics. His research areas include information technology, server, and database. He can be contacted at email: hiro@ittelkom-pwt.ac.id.