Net's success on segmentation tasks has been argued to come from its explicit extraction of features at multiple resolutions, which allows it to respond to fine-grained features as well as long-range dependencies in the input. In representation learning for audio, we are interested in constructing encodings which similarly carry both high- and low-frequency information. Because U-Nets were originally designed for image segmentation, however, variants of the architecture traditionally produce an output at the spatial resolution of the input: $\mathbb{R}^{I \times C}$ where I is the input length and C the number of classes given by the task. Representation learning models, on the other hand, aim to decouple the dimensionality of the representation from the temporal resolution of the original input signal, to accommodate usage in a wide variety of downstream tasks. They typically output a single representation: $\mathbb{R}^{R}$ with R the size of the representation. To address this mismatch, we propose adding a tail module to the U-Net. The tail module serves (1) to combine enriched sequence length representation obtained at multiple resolutions in the contractive and expansive path, and (2) to reduce the resolution of these features to a single compact multi-timescale representation. See Figure 1 for an overview of our architecture.

In addition to the tail module, we introduce another improvement on the TUNe architectures: TUNe+. TUNe+ networks have additional connections between the expansive and tail path (the pink arrows and blocks in Figure 1). This explicit transfer of information at lower resolutions should allow for better multi-timescale feature information flow, further improving the obtained representations.

3.1 Contractive Path

The contractive path of the network is built from a block consisting of a convolution, a batch normalisation layer, and a ReLU activation function, following CLMR's encoding architecture [3]. This block is repeated N_{con} times (default 4). In Figure 1, this block is indicated with purple arrows. The output of this block is saved to combine with the corresponding expansive block later. After each block, max pooling is applied to extract the most prominent patterns at a given resolution, represented by the red arrows in Figure 1. The output of the max pooling is then passed on either to the next layer in the contractive path or, after the N_{con}-th block, to the beginning of the expansive path. Every layer in the encoder path doubles the amount of channels. The convolution operation is performed with a kernel size of 3 and a stride of 1, and the max pooling operation with a kernel size of 3 and stride of 3. Furthermore, the convolution operation is applied on a padded input, such that input and output to the convolution are of equal sequence length.

3.2 Expansive Path

Whereas each layer of the contractive path reduces the signal resolution, the expansive path of the network is made up of blocks that are paired with blocks from the contractive path, each one gradually *increasing* the signal resolution again. The expansive path block, which is repeated N_{exp} times (default 4), consists of a strided transposed convolution, represented by the green arrows in Figure 1, with a kernel size of 3, stride of 3, and 0 padding, to keep the dimensionality the same as the output of the contractive block at the same depth. The output of each strided transposed convolution is concatenated with the output of its pair from the contractive path (see the blue arrows leading into the blue-and-white blocks in Figure 1). Next, a convolution followed by batch normalisation and ReLU activation is used to combine the multi-scale features in the concatenated outputs. The upsampling by strided transposed convolution enables the network to combine lower frequency features with the higher frequency information obtained at higher resolutions in the contractive path. The input–output channel ratio for the strided transposed convolution is 2:1, and for the convolution block is also 2:1, because of the concatenation of the encoder block and strided transposed convolution output.

3.3 Tail

The blocks in the tail of a TUNe model are identical to the blocks of the contractive path, each of the N_{tai} blocks (default 4) doubling the number of channels until the representation size is reached. In a TUNe+ model, the tail blocks differ from the contractive path blocks because TUNe+ tail blocks are paired with blocks from the expansive path. As illustrated by the pink arrows in Figure 1, each TUNe+ tail block combines the output of the previous tail block with the output of the expansive block at the same resolution. The input–output channel ratio in this case is 3:2, because the input gets ⅔ of the input channels from the expansive path and ⅓ from the previous tail block. Depending on the number of downsampling steps in the tail module, any remaining temporal dimensions O are projected to a single representation using average pooling, mapping from $\mathbb{R}^{O \times R}$ to $\mathbb{R}^{R}$ with R the representation size. This operation is indicated by the yellow arrow in Figure 1.

The TUNe network architecture allows for flexible network adjustments. When removing all tail layers the resulting TUNe is equivalent to the original U-Net. If all expansive layers are removed there is no way to distinguish contractive and tail layers, and adding six contractive or tail layers is equivalent to CLMR. Since the output does not have to end with the same dimensionality as the input, one can add and remove contractive, expansive, or tail layers without needing to add or remove layers along the other paths. We leverage this compositionality to conduct experiments with the TUNe architecture that explore representations built up over a differing number of timescales, in order to explore the impact of the structure of each respective module on the capacity of the resulting representation to perform in downstream tasks.

4. EXPERIMENTS

To explore the validity and performance of our proposed setup, we trained multiple TUNe architectures with varying path lengths for 1000 epochs on the MagnaTagATune

Variant	N_{con}	N_{exp}	N_{tai}	Filters	Parameters (M)	MTT_{AUC}	MTT_{AP}
Vanilla TUNe	4	4	4	34	2.4	87.7	33.0
TUNe Contractive+1	5	5	4	18	2.3	88.3	33.9
TUNe Contractive+2	6	6	4	9	2.1	88.6	34.6
TUNe Contractive+3	7	7	4	4	1.7	88.0	33.5
TUNe Expansive-1	4	3	3	34	2.3	87.6	33.0
TUNe Expansive-2	4	2	2	35	2.4	87.7	33.1
TUNe Expansive-3	4	1	1	38	2.3	87.6	33.0
TUNe Tail+1	4	4	5	28	2.3	88.2	34.4
TUNe Tail+2	4	4	6	19	2.3	88.7	35.2
TUNe Tail+3	4	4	7	15	2.3	89.1	36.5
TUNe Tail+4	4	4	8	13	2.3	89.2	36.6
TUNe Tail+5	4	4	9	11	2.1	89.2	36.5
TUNe CLMR-tail	4	4	10*	10*	2.5	89.4	36.7
TUNe+	4	4	9	11	2.2	89.2	36.6
Vanilla TUNe Small	4	4	4	11	0.4	86.8	31.9
TUNe+ Large	4	4	9	34	7.4	89.4	37.1
TUNe+ Smaller Rep	4	4	9	11	1.4	89.2	36.1
musicnn 10 000 [2]	-	-	-	–	11.8**	90.7	38.4
TUNe Tail+5 10 000	**4**	**4**	**9**	**11**	**2.1**	**89.5 (89.6)**	**37.0 (36.7)**
TUNe+ 10 000	**4**	**4**	**9**	**11**	**2.2**	**89.3 (89.8)**	**37.1 (37.1)**
CLMR 10 000 [3]	10	0	0	–	2.4	88.7 (89.3)	35.6 (36.0)

Table 1: TUNe variant performance on the MagnaTagATune (MTT) tag prediction task with number of parameters and number of initial filters. The performance is measured after 1000 epochs (except where noted otherwise) with the area under the receiver operating characteristic curve MTT_{AUC}, and the average precision, MTT_{AP}. The table is divided into six sections: the Vanilla TUNe model with no layers added; the results of adding contractive layers; the results of removing expansive layers; the results of adding tail layers; the results parameter-efficiency experiments; and the results of the best TUNe models and the CLMR baseline after training for 10 000 epochs. In addition to the shallow probe, we trained a probe with an extra linear layer for the longer trained models and report this score in parentheses. TUNe Tail+5 and TUNe+ at 10 000 epochs are the best-performing models overall, exceeding CLMR's performance at 10 000 epochs and performing only slightly worse than state-of-the-art end-to-end–trained musicnn. Multiple other variants match CLMR performance even though only trained for 1 000 epochs.

* For the CLMR-tail experiment, the number of filters corresponds to the initial number of filters used for the contractive and expansive path. For the tail, we used the same architecture as CLMR's SampleCNN and report the number of blocks.

** Number of parameters is taken from the last reported number of parameters, musicnn [31].

audio dataset. As baseline, we compare to CLMR [3]. We ensure a fair comparison by varying the number of channels in each TUNe architecture to obtain a total parameter count comparable to the CLMR baseline. Next, we trained five variants: (1) a version where the tail was fixed to the published pre-trained CLMR model, in order to test whether the contractive and expansive paths (the 'U') add information; (2) a TUNe+ network, still restricted to have no more parameters than CLMR, to test whether the extra connections between the expansive path and the tail allow for better feature information flow; (3) a filter-restricted model, to test whether the number of filters can be a bottleneck for the deeper models; (4) a large TUNe+ network with an unrestricted number of parameters; and (5) a model with a smaller representation dimension. Finally, we evaluated the out-of-domain dataset generalisability of our two best models on three different datasets for the same probing task. [1]

[1] Pretrained models and source code for all experiments are available to download at `https://github.com/Marcel-Velez/TUNe`

4.1 Hyperparameters and Preprocessing

The hyperparameters we used for pre-training were the same as [3], following their setup with data augmentations. We used an Adam optimiser [32], and He initialisation [33] for all convolutional layers. As input, we sampled audio files at 22 050 Hz for 59 049 samples. We also used the same architecture for the projector head as [3]. This output is then used for the contrastive learning objective. For every experiment, we used a batch size of 96. In order to be able to train with such a batch size, we trained every model data-parallel (DP) on two Titan RTXs, except for Vanilla TUNe and TUNe+ Large. To train these variants with a batch size of 96, we used three Titan RTXs.

4.2 Exploring the TUNe Architecture

Because the number of permutations of how many contractive layers are added, expansive layers are subtracted, tail layers are added grows exponentially, we chose to investigate the influence of each of these paths separately. Every model was trained on the audio of the MagnaTagA-

Tune (MTT) dataset [34]. This dataset consists of 25 863 music clips of 29 seconds of audio from 5223 songs. Each of these clips has one or more tags, making it a multi-class classification task dataset. We use the same train-validation-test split as is common within MIR [35–37]. After training, each model was probed on the MTT tagging task using a single linear-layer probe and evaluated using two metrics. The first metric is the receiver operating characteristic curve (MTT_{AUC}), which is popular but can be positively biased for imbalanced datasets [38], and the average precision (MTT_{AP}). We report results for each model in Table 1.

The only constraint for these variations was to have fewer parameters than CLMR, so as to exclude the model being bigger being a possible reason for performing better. The TUNe architecture follows a fixed input channel–output channel ratio per layer, and thus in order to keep the number of parameters smaller than CLMR, we only had to change the number of output channels of the first block. The remainder of the network changes in proportion. A complete overview of the number of parameters and initial output filters can be found in Table 1.

4.2.1 *Contractive Path Depth*

When varying the contractive path depth, we also added an equal number of expansive layers, in order to keep the upper resolution of the U and the tail the same; the tail remained unchanged so that the final representation size remained unchanged. Starting from the vanilla default of 4 contractive layers, adding layers increased performance up until $N_{\text{con}} = 6$, suggesting that the extra contractive and expansive layers – a deeper U – do allow for better integration of feature information for the representation dimensionality. Contractive+3 ($N_{\text{con}} = 7$) drops slightly in performance, which initially seems contradictory to the Contractive+1 and +2 results. We believe this can be attributed to the potentially exponential parameter growth of adding layers to the contractive and thus also expansive paths: to prevent the exponential growth and remain within our constraint on the maximum number of parameters, adding layers entails reducing the number of filters. TUNe Contractive+3, for example, has only 4 initial filters, as compared to 34 in Vanilla TUNe, and this may no longer be enough to achieve good performance.

To test this hypothesis, we ran Vanilla TUNe with the number of filters from TUNe+ (labelled Vanilla TUNe Small in Table 1), and conversely ran TUNe+ with the number of filters from Vanilla TUNe (labelled TUNe+ Large in Table 1). Vanilla TUNe performance dropped and TUNe+ Large performance increased under these conditions, suggesting that the number filters could indeed be the bottleneck for Contractive+3.

4.2.2 *Expansive Path Height*

In order to analyze the effect of expansive path height, we applied a similar procedure. For every expansive layer we removed, we also removed a tail layer to keep the final representation dimensionality unchanged. Because this modification reduced dimensionality, we were able to *add* initial filters in order to come closer to the dimensionality of CLMR. Nonetheless, removing expansive layers seems to have little influence: removing up to three layers of the expansive path and tail leaves performance essentially unchanged. Put differently, the extra initial filters seem to be able to compensate for reduced integration of the higher frequency timescale features due to a shallower U.

To see whether the U shape in fact adds information and increases performance, we trained a TUNe model with a tail path identical to pretrained CLMR. If the U shape does not add information, such a model should perform equally well or worse than baseline CLMR; if it performs better, then there is evidence that the contractive and expansive paths are contributing signal enrichment important for representation learning. Indeed, TUNe with a CLMR tail performs better with a single layer probe than baseline CLMR can even after 10 times as many training epochs and a multi-layer probe. It seems that the contractive–expansive path pair is a powerful performance enhancer for time-domain music representation learning.

4.2.3 *Tail Length*

The model performance after adding 1 to 5 layers to the tail path shows a steady increase per layer added until Tail+4, after which performance seems to plateau. In general, we should expect longer tails to improve performance, because the model average pools the remaining sequence length at the end of the tail path. With fewer tail layers, we average over a longer sequence, and when averaging, detailed information is replaced with an aggregation, thereby losing possible important information. The parameter constraint could again be responsible for the eventual plateau, as we see that the TUNe+ Large model from before also performs better than either Tail+4 or Tail+5. But it is a plateau, not a decrease: Tail+5 performs equivalently to Tail+4, and it does so with fewer parameters and potentially less loss of precision from averaging than Tail+4. We choose Tail+5 as our best model from this block.

4.2.4 *Extra Connections*

The model with extra connections, TUNe+, performs comparably to the Tail+5 model, with only .1 higher MTT_{AP}. It still shows similar MTT_{AUC} scores as CLMR 10 000 epochs trained and outperforms CLMR MTT_{AP}-wise. To further explore the influence of the added connections we chose TUNe+ to be one of the two variants used for the probing experiments (Section 4.3).

4.2.5 *Smaller Representations*

All of these variants showed reasonable performance for the same representation size, with only slightly varying numbers of parameters. To test the parameter efficiency of TUNe architectures, we trained another variant of the TUNe+ with the same number of initial filters, but reducing the representation size from 512 to 256. The results were impressive. Even with 44% fewer parameters and a representation size half that of the other TUNe architectures we tested, TUNe+ Smaller Rep still performs equally well as CLMR after 10 000 epochs. Compared to the best TUNe architectures, it achieves comparable MTT_{AUC} and only marginally worse MTT_{AP}.

4.2.6 Longer Training

In order to be conservative with computing resources, we first trained all of the aforementioned models for 1000 epochs only and probed with only a single linear layer. As a final comparison, we trained the two best models, TUNe Tail+5 and TUNe+, an additional 9 000 epochs. Because of the random data augmentations and the size of MTT, training for more epochs results in the models 'hearing' an increasing amount of natural variation in the audio, which in turn improves performance. We evaluated these models using the same probing tasks, once with a single linear layer and once with a two-layer probe to see how much introducing nonlinearity could increase performance. In Table 1, we report these additional multi-layer performance figures in brackets. The MTT_{AUC} score does increase with the multilayer probe, but the MTT_{AP}, on the contrary, decreases. Overall, however, these two 10 000-epoch models are the best performing models from our entire series of experiments, achieving slightly lower performance than the musically motivated end-to-end trained musicnn [2] and outperforming CLMR 10 000 epoch results.

4.3 Probing Tasks

In order to compare the TUNe performance to state-of-the-art models, we evaluate TUNe with probing: training a shallow model on downstream tasks [10]. We use the same datasets as [3] for the CLMR training.

When probing a model, we evaluate the pre-trained model representations on a different dataset than the model is pre-trained on. This evaluation on a different dataset is done by training a probe, often a single linear layer or a multi-layer perceptron with one hidden layer. This probe takes the output representation of the main model as input and outputs the desired classes or values for the task in question. Probing is often used to test certain representation characteristics, in our case, to see how well the learned representations from other types of datasets generalise to music tagging.

We ran the probing experiment to see how well our network could generalise representations when trained on three different datasets: the medium Free Music Archive dataset [39], the fault-filtered GTZAN dataset [40,41] containing 930 songs, and the McGill Billboard dataset [42] containing 712 songs. Next, we probed the trained models on the MTT dataset, of which the results are displayed in Table 2. For this probing experiment, we used a learning rate of 3e^{-4}, weight decay of 10^{-6}, and an early stopping mechanism. Early stopping occurred if the probe's validation score did not improve for five epochs.

TUNe architectures allow for excellent out-of-dataset representation generalisability. Even when both TUNe Tail+5 and TUNe+ were trained on the McGill Billboard dataset, which is more than 33 times smaller than MTT, the models still only perform 4% worse on AUC than the models pre-trained on MTT. When they are trained on the GTZAN dataset, which is about the same size as McGill Billboard, both variants outperform CLMR regardless of (non-MTT) training set. With pre-training on the MTT-sized FMA dataset, the TUNe models perform almost as well as they did in the original MTT-only experiments.

Probing Variant	Training Data	MTT_{AUC}	MTT_{AP}
TUNe+	FMA	89.1 (89.4)	36.2 (36.1)
TUNe Tail +5	FMA	88.9 (89.2)	35.3 (35.9)
CLMR	FMA	86.2 (86.6)	30.6 (31.2)
TUNe+	GTZAN	87.2 (87.9)	32.6 (33.9)
TUNe Tail +5	GTZAN	86.9 (87.5)	32.6 (33.0)
CLMR	GTZAN	81.9 (85.4)	26.2 (29.5)
TUNe Tail +5	Billboard	84.7 (85.8)	28.6 (29.9)
TUNe+	Billboard	84.5 (85.9)	28.7 (30.5)
CLMR	Billboard	82.7 (84.2)	26.9 (27.8)

Table 2: TUNe Tail +5, TUNe+ and CLMR out-of-domain probing experiments. The table shows the probing performance on MagnaTagATune of each of the three models, trained on the Free Music Archive medium (FMA), fault-filtered GTZAN, and the McGill Billboard dataset. In addition to the shallow probe, we trained a probe with an extra linear layer and report this score in parentheses. CLMR results are taken from [3].

5. CONCLUSION

In this paper, we introduced TUNe network architectures, a generalisation of a recent representation learning framework called CLMR. TUNe brings the strengths of U-Nets to representation learning. TUNe networks comprise three sections, called the contractive, expansive, and tail paths, which can be flexibly lengthened or shortened. We performed several experiments exploring the contribution of each of these paths and compared them against CLMR. We evaluated TUNe's performance in three ways. First, we trained and evaluated variants of our model with CLMR on MagnaTagATune (MTT), outperforming CLMR marginally after training for a fraction of CLMR's time. Second, we evaluated the best two models with out-of-domain probing tasks. Both TUNe architectures improved upon the already competitive CLMR performance and showed that TUNe architectures allow for an even better generalisation of music representations. In the supplemental material, we include the results of further experiments showing how TUNe architectures can also achieve competitive performance on other downstream tasks, even at small model sizes. TUNe sets a new standard for parameter efficiency and the ability of modern self-supervised networks to extract salient features, and we hope that it will encourage MIR researchers to use self-supervised music representations more widely.

6. REFERENCES

[1] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE*

transactions on pattern analysis and machine intelligence, vol. 35, no. 8, pp. 1798–1828, 2013.

[2] J. Pons and X. Serra, "musicnn: Pre-trained convolutional neural networks for music audio tagging," 2019.

[3] J. Spijkervet and J. A. Burgoyne, "Contrastive learning of musical representations," in *Proceedings of the 22nd International Society for Music Information Retrieval Conference*, 2021.

[4] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 449–12 460, 2020.

[5] D. Hupkes, S. Veldhoen, and W. Zuidema, "Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure," *Journal of Artificial Intelligence Research*, vol. 61, pp. 907–926, 2018.

[6] A. Conneau, G. Kruszewski, G. Lample, L. Barrault, and M. Baroni, "What you can cram into a single vector: Probing sentence embeddings for linguistic properties," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2018, pp. 2126–2136. [Online]. Available: https://aclanthology.org/P18-1198

[7] J. Hewitt and C. D. Manning, "A structural probe for finding syntax in word representations," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1, 2019, pp. 4129–4138.

[8] A. Van Den Oord, S. Dieleman, and B. Schrauwen, "Transfer learning by supervised pre-training for audio-based music classification," in *Proceedings of the 15th International Society for Music Information Retrieval Conference*, 2014.

[9] P. Hamel, M. E. Davies, K. Yoshii, and M. Goto, "Transfer learning in MIR: Sharing learned latent representations for music audio classification and similarity," in *Proceedings of the 14th International Society for Music Information Retrieval Conference*, 2013.

[10] K. Choi, G. Fazekas, M. B. Sandler, and K. Cho, "Transfer learning for music classification and regression tasks," in *Proceedings of the 18th International Society for Music Information Retrieval Conference*, 2017.

[11] J. Lee, J. Park, and J. Nam, "Representation learning of music using artist, album, and track information," Paper presented at the Machine Learning for Music Discovery Workshop, International Conference on Machine Learning, 2019.

[12] Q. Huang, A. Jansen, L. Zhang, D. P. Ellis, R. A. Saurous, and J. Anderson, "Large-scale weakly-supervised content embeddings for music recommendation and tagging," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2020, pp. 8364–8368.

[13] J. Kim, J. Urbano, C. Liem, and A. Hanjalic, "One deep music representation to rule them all? A comparative analysis of different representation learning strategies," *Neural Computing and Applications*, vol. 32, no. 4, pp. 1067–1093, 2020.

[14] R. Castellon, C. Donahue, and P. Liang, "Codified audio language modeling learns useful representations for music information retrieval," in *Proceedings of the 22nd International Society for Music Information Retrieval Conference*, 2021.

[15] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," arXiv:2111.06377, 2021.

[16] B. Zhang, J. Leitner, and S. Thornton, "Audio recognition using mel spectrograms and convolution neural networks," Online report, Noiselab, University of California, 2019.

[17] J. S. Gómez, J. Abeßer, and E. Cano, "Jazz solo instrument classification with convolutional neural networks, source separation, and transfer learning," in *Proceedings of the 19th International Society for Music Information Retrieval Conference*, 2018, pp. 577–584.

[18] M. A. Rohit, A. Bhattacharjee, and P. Rao, "Four-way classification of tabla strokes with models adapted from automatic drum transcription," in *Proceedings of the 22nd International Society for Music Information Retrieval Conference*, 2021.

[19] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 234–241.

[21] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[22] A. Jansson, E. Humphrey, N. Montecchio, R. Bittner, A. Kumar, and T. Weyde, "Singing voice separation with deep U-Net convolutional networks," in *Proceedings of the 18th International Society for Music Information Retrieval Conference*, 2017, pp. 23–27.

[23] D. Stoller, S. Ewert, and S. Dixon, "Wave-U-Net: A multi-scale neural network for end-to-end audio source separation," in *Proceedings of the 19th International Society for Music Information Retrieval Conference*, 2018.

[24] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," Paper presented at the Speech Synthesis Workshop, International Speech Communication Association, 2016.

[25] D. Stoller, M. Tian, S. Ewert, and S. Dixon, "Seq-U-Net: A one-dimensional causal U-Net for efficient sequence modelling," in *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, 2020, pp. 2893–2900.

[26] L. Wiskott and T. J. Sejnowski, "Slow feature analysis: Unsupervised learning of invariances," *Neural Computation*, vol. 14, no. 4, pp. 715–770, 2002.

[27] J. Lee, J. Park, K. L. Kim, and J. Nam, "SampleCNN: End-to-end deep convolutional neural networks using very small filters for music classification," *Applied Sciences*, vol. 8, no. 1, p. 150, 2018.

[28] M. Lederle and B. Wilhelm, "Combining high-level features of raw audio waves and mel-spectrograms for audio tagging," Submission to the Detection and Classification of Acoustic Scenes and Events challenge, 2018.

[29] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," arXiv:2005.00341, 2020.

[30] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning*, 2020, pp. 1597–1607.

[31] J. Pons, O. Nieto, M. Prockup, E. M. Schmidt, A. F. Ehmann, and X. Serra, "End-to-end learning for music audio tagging at scale," in *Proceedings of the 19th International Society for Music Information Retrieval Conference*, 2018.

[32] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations*, 2014.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[34] E. Law, K. West, M. I. Mandel, M. Bay, and J. S. Downie, "Evaluation of algorithms using games: The case of music tagging," in *Proceedings of the 10th International Society for Music Information Retrieval Conference*, 2009, pp. 387–392.

[35] J. Pons, O. Nieto, M. Prockup, E. M. Schmidt, A. F. Ehmann, and X. Serra, "End-to-end learning for music audio tagging at scale," in *Proceedings of the 19th International Society for Music Information Retrieval Conference*, 2018.

[36] S. Dieleman and B. Schrauwen, "Multiscale approaches to music audio feature learning," in *Proceedings of the 14th International Society for Music Information Retrieval Conference*, 2013, pp. 116–121.

[37] M. Tagliasacchi, B. Gfeller, F. de Chaumont Quitry, and D. Roblek, "Pre-training audio representations with self-supervision," *IEEE Signal Processing Letters*, vol. 27, pp. 600–604, 2020.

[38] J. Davis and M. Goadrich, "The relationship between precision–recall and ROC curves," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 233–240.

[39] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson, "FMA: A dataset for music analysis," in *Proceedings of the 18th International Society for Music Information Retrieval Conference*, 2017.

[40] B. L. Sturm, "The GTZAN dataset: Its contents, its faults, their effects on evaluation, and its future use," arXiv:1306.1461, 2013.

[41] C. Kereliuk, B. L. Sturm, and J. Larsen, "Deep learning and music adversaries," *IEEE Transactions on Multimedia*, vol. 17, no. 11, pp. 2059–2071, 2015.

[42] J. A. Burgoyne, J. Wild, and I. Fujinaga, "An expert ground truth set for audio chord recognition and music analysis," in *Proceedings of the 12th International Society for Music Information Retrieval Conference*, 2011, pp. 633–638.

[43] J. Spijkervet, "Contrastive Learning of Musical Representations," Master's thesis, University of Amsterdam, 2021.

[44] F. Weninger, F. Eyben, and B. Schuller, "On-line continuous-time music mood regression with deep recurrent neural networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014, pp. 5412–5416.

[45] E. Koh and S. Dubnov, "Comparison and analysis of deep audio embeddings for music emotion recognition," arXiv:2104.06517, 2021.

[46] Pioneer, "Rekordbox v3.2.2," 2015, available online at http://www.cp.jku.at/datasets/giantsteps/.

[47] J. Jiang, G. G. Xia, and D. B. Carlton, "Crowd annotation for audio key estimation," 2019.

[48] F. Medhat, D. Chesmore, and J. Robinson, “Masked conditional neural networks for audio classification,” in *Proceedings of the International Conference on Artificial Neural Networks*, 2017, pp. 349–358.

[49] F. Korzeniowski and G. Widmer, “End-to-end musical key estimation using a convolutional neural network,” in *Proceedings of the 25th European Signal Processing Conference*, 2017, pp. 966–970.

[50] G. Tzanetakis and P. Cook, “Musical genre classification of audio signals,” *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.

[51] P. Knees, Á. Faraldo Pérez, H. Boyer, R. Vogl, S. Böck, F. Hörschläger, and M. Le Goff, “Two data sets for tempo estimation and key detection in electronic dance music annotated from user corrections,” in *Proceedings of the 16th International Society for Music Information Retrieval Conference*, 2015.

[52] M. Soleymani, M. N. Caro, E. M. Schmidt, C.-Y. Sha, and Y.-H. Yang, “1000 songs for emotional analysis of music,” in *Proceedings of the 2nd ACM International Workshop on Crowdsourcing for Multimedia*, 2013, pp. 1–6.

DDSP-BASED SINGING VOCODERS: A NEW SUBTRACTIVE-BASED SYNTHESIZER AND A COMPREHENSIVE EVALUATION

Da-Yi Wu[1*] **Wen-Yi Hsiao**[2*] **Fu-Rong Yang**[3*] **Oscar Friedman**[4]
Warren Jackson[5] **Scott Bruzenak**[4] **Yi-Wen Liu**[3] **Yi-Hsuan Yang**[1,2]

[1] Academia Sinica, [2] Taiwan AI Labs, [3] National Tsing Hua Univ., [4] 470 Music Group, [5] PARC
{ericwudayi2, s101062219, fjbcrs34}@gmail.com

ABSTRACT

A vocoder is a conditional audio generation model that converts acoustic features such as mel-spectrograms into waveforms. Taking inspiration from Differentiable Digital Signal Processing (DDSP), we propose a new vocoder named SawSing for singing voices. SawSing synthesizes the harmonic part of singing voices by filtering a sawtooth source signal with a linear time-variant finite impulse response filter whose coefficients are estimated from the input mel-spectrogram by a neural network. As this approach enforces phase continuity, SawSing can generate singing voices without the phase-discontinuity glitch of many existing vocoders. Moreover, the source-filter assumption provides an inductive bias that allows SawSing to be trained on a small amount of data. Our evaluation shows that SawSing converges much faster and outperforms state-of-the-art generative adversarial network- and diffusion-based vocoders in a resource-limited scenario with only 3 training recordings and a 3-hour training time.[*]

1. INTRODUCTION

Singing voice synthesis (SVS) aims to generate human-like singing voices from musical scores with lyrics [1–9]. State-of-the-art (SOTA) voice synthesis techniques involve two stages: acoustic feature modeling from musical scores and audio sample reconstruction via a so-called "vocoder." A *neural vocoder* takes an acoustic feature such as mel-spectrogram as input and outputs a waveform using deep learning networks [10–22]. However, phase discontinuities within partials often occur due to the difficulty of reconstructing realistic phase information from a mel-spectrogram. This may lead to a short-duration broadband transient perceived as "glitch" or "voice tremor," which is more audible during long utterances commonly found in singing [18], as exemplified in Figure 1.

[*]Equal contribution. Preliminary work was done while Wu was a remote intern working with Friedman at 470 Music Group, LLC.

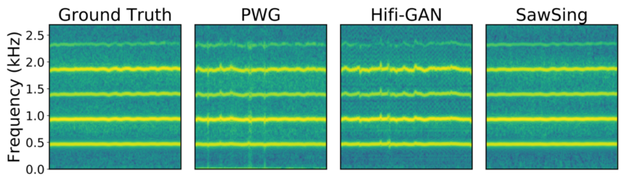

Figure 1: The magnitude spectrograms of a long utterance of an original recording ('ground truth') and those reconstructed by two widely-used neural vocoders, Parallel WaveGAN (PWG) [14] and HiFi-GAN [15], and the proposed SawSing. Each vocoder is trained on 3 hours of recordings from a female singer until convergence. We see glitches in the results of PWG and HiFi-GAN.

Differentiable Digital Signal Processing (DDSP) [23] introduces a new paradigm for neural audio synthesis. It incorporates classical digital signal processing (DSP) synthesizers and effects as differentiable functions within a neural network (NN), and combines the expressiveness of an NN with the interpretability of classical DSP. The use of phase-continuous oscillators is a potential solution to the phase problem from which regular neural vocoders suffer, and the strong inductive bias of this approach may obviate the need of large training data. Furthermore, DDSP has already succeeded in achieving sound synthesis of, and timbre transfer between, monophonic instruments [23–30]. These motivate us to explore whether the DDSP approach can be applied to build a singing vocoder.

This paper proposes SawSing, a DDSP-based singing vocoder which reconstructs a monophonic singing voice from a mel-spectrogram. The architecture of SawSing similarly consists of an NN and classical DSP components; unlike DDSP, its DSP portion is a subtractive harmonic synthesizer which filters a sawtooth waveform containing all possible harmonic partials, plus a subtractive noise synthesizer which filters uniform noise. The sawtooth signal enforces phase continuity *within* partials, thereby avoiding the glitches. Moreover, the partials of a sawtooth signal are guaranteed to be in phase, so it also enforces the phase coherence *between* partials, intrinsic to human voices. The function of the NN, on the other hand, is to infer from the mel-spectrogram the fundamental frequency (f0) of the sawtooth signal and the filter coefficients of the harmonic and noise synthesizers for each time frame.

In our experiments, we use data from two singers (each three hours) of the MPop600 Mandarin singing corpus [31]. We compare the performance of SawSing with the neural source-filter (NSF) model [12], two existing DDSP-based synthesizers, the original additive-based DDSP [23] and the differentiable wavetable synthesizer [27], and a few famous neural vocoders, i.e., two generative adversarial network (GAN)-based models [5, 14] and a diffusion-based model [21]. We consider both a regular scenario where the vocoders are trained for days using the 3-hour dataset, and a resource-limited scenario with constraints on training data and training time. Our experiments show that SawSing converges much faster and outperforms the other vocoders in the resource-limited scenario.

The main contribution of the paper is two-fold. First, we show that despite differences between instrumental sounds and singing voices [32], the classic idea of subtractive synthesis [33, 34] can be applied to singing voices using the DDSP approach.[1] Second, we provide empirical evidences showing that DDSP-based vocoders can compare favorably with sophisticated, SOTA neural vocoders. Furthermore, since DDSP-based vocoders are lightweight and training-efficient, they have the potential to be used in creative and real-time scenarios of singing expression with limited training data of a target singing voice [36, 37].

We open source our code at `https://github.com/YatingMusic/ddsp-singing-vocoders/`. For audio examples, visit our demo webpage `https://ddspvocoder.github.io/ismir-demo/`.

2. BACKGROUND

Neural vocoders often aim to reconstruct a waveform $\mathbf{y} \in \mathbb{R}^{1\times T}$ from an input mel-spectrogram $\mathbf{X} \in \mathbb{R}^{M\times N}$:

$$\mathbf{y} = f_{\text{vocoder}}(\mathbf{X}), \tag{1}$$

where M, N, T denote respectively the number of mel filter banks, spectral frames, and time-domain samples. The conversion from $\mathbf{X}$ to $\mathbf{y}$ can be done, for example, by upsampling $\mathbf{X}$ multiple times through transposed *convolutions* until the length of the output sequence matches the temporal resolution of the raw waveform [13, 15]. As usual reconstruction loss functions such as mean-square errors cannot reflect the perceptual quality of the reconstruction, GAN-based approaches [13–15] learn discriminators to better guide the learning process of the generator (i.e., f_{vocoder}). Newer diffusion-based approaches [20, 21] avoid the use of discriminators and learn to convert white Gaussian noises $\mathbf{z} \in \mathbb{R}^{1\times T}$ (i.e., of the same length as $\mathbf{y}$) into structured waveform $\mathbf{y}$ through a denoising-like Markov chain, using $\mathbf{X}$ as a condition. The mapping process between $\mathbf{X}$ and $\mathbf{y}$ of such neural vocoders appears to be a black box that is hard to interpret. However, given sufficient training data (e.g., recordings amounting to 24 hours [15, 20, 21, 38] or 80 hours [18]) and training time (e.g., days), SOTA neural vocoders can reconstruct the waveforms with high fidelity.

The majority of neural vocoders, however, have been originally developed for speech. When the rate of utterances is fast, as is common in speech, the glitches resulting from the phase discontinuities within partials may be perceptually masked by the natural transients of the voice. However, during singing, where long utterances are common, these discontinuities are more audible.

To improve the performance of GAN-based vocoders for singing voices, the idea of incorporating the f0 information has been explored recently. PeriodNet [16] uses f0 to create sine excitation as input to Parallel WaveGAN (PWG) [14] to model the periodic part of human voices. Guo *et al.* [17] further filter such an f0-driven excitation signal with a linear time-variant finite impulse response (LTV-FIR) filter whose coefficients are estimated from the input mel-spectrogram, and use the resulting "harmonic signal" as input to PWG and MelGAN [13]. SingGAN [18] uses more complicated "adaptive feature learning" layers to incorporate the f0. These models were shown to outperform older GAN-based vocoders such as PWG and MelGAN in listening tests, but no evaluations against the newest GAN-based vocoder HiFi-GAN [15] and diffusion-based vocoders were reported. Moreover, their evaluation did not consider resource-limited scenarios.

We propose in this paper a radically different approach that uses traditional DSP synthesizers (instead of upsampling convolutions) as the backbone for f_{vocoder}. While the ideas in DDSP have flourished and been applied to synthesizing not only instrumental sounds [23–27], but also audio effects [39–43], their application to singing synthesis remains under-explored. The only exception, to our knowledge, is the preliminary work presented by Alonso and Erkut [44], which employed exactly the same additive synthesizer as the original DDSP paper [23]. However, they did not compare the performance of their vocoder with any other vocoders. Our work extends theirs by using a subtractive harmonic synthesizer instead, with comprehensive performance evaluations against SOTA neural vocoders such as HiFi-GAN and FastDiff [21].

We note that, while a DDSP-based vocoder may solve the glitch problems by inducing continuous phase hypothesis using a harmonic synthesizer, this hypothesis may constrain the model learning ability. Experiments reported in this paper are needed to study its performance.

Publicly-available training corpora for singing tend to be much smaller than those for speech [31, 45] (often $\leq$10 hours). Therefore, besides tackling the glitch problem, our premise is that SawSing can learn faster than prevalent neural vocoders without a large training corpus, due to its strong inductive bias. Moreover, the success of SawSing may pave the way for the exploration of other advanced DSP components for singing synthesis in the future.

3. ORIGINAL DDSP-ADD SYNTHESIZER

The idea of DDSP is to use DSP synthesizers to synthesize the target audio, with the parameters of the synthesizers Φ

[1] We note that the use of a sawtooth waveform in DDSP-based models has been attempted for speech synthesis [35] and instrumental synthesizer sound matching [26], but its application to singing vocoder is new.