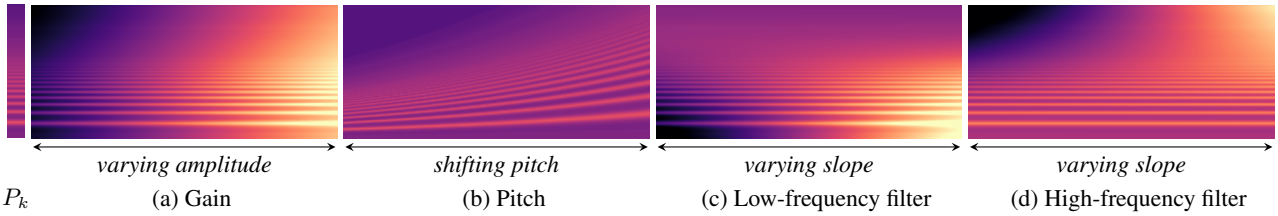



**Figure 3**: **Spectral Transformations.** Our proposed transformations learn to alter the gain (a), pitch (b) and frequency support (c,d) of a given prototype spectrogram $P_k$.

We use linear interpolation for handling non-integer shifts.

• *High- and Low-Frequency Filter.* Given an audio sample $x$, we define for each time $t$ an affine low-frequency filter $\mathbf{L}_k(x)[t] \in \mathbf{R}^F$ by predicting its slope and cutoff frequency. Similarly, we predict a high-frequency filter $\mathbf{H}_k(x)[t] \in \mathbb{R}^F$. The transformations $\mathcal{T}_L^{\text{low}}$ and $\mathcal{T}_H^{\text{high}}$ simply add the corresponding filters $L$ and $H$ to a log-Mel spectrogram.

These transformations can be written as follows for a log-Mel spectrogram $m \in \mathbb{R}^F$ and a mel-frequency $f \in [1, F]$:

$$\mathcal{T}_g^{\text{gain}}(m)[f] = m[f] + g \tag{1}$$
$$\mathcal{T}_s^{\text{pitch}}(m)[f] = m[\Phi(f,s)] \tag{2}$$
$$\mathcal{T}_L^{\text{low}}(m)[f] = m[f] + L[f] \tag{3}$$
$$\mathcal{T}_H^{\text{high}}(m)[f] = m[f] + H[f]\,, \tag{4}$$

with $\Phi(f,s) = A\log_{10}\left(1 + s\left(10^{f/A} - 1\right)\right)$ the mel-aware pitch shifting, with $A = 2595$ [19]. Our transformations have links with moment matching methods [20, 21]. In fact, modulating the amplitude, pitch, and spectral support of spectrograms is similar to matching the moment of zeroth order (amplitude gain), first order (pitch shift), and second order (frequency filters).

**Reconstruction Model.** For each prototype $k$ and each time step $t$, we successively apply the gain, pitch shift, low-pass, and high-pass transformations in this specific order. The resulting reconstruction $\mathcal{R}_k(x)$ of an input sound $x$ by a prototype $P_k$ is defined as follows:

$$\mathcal{R}_k(x)[t] = \mathcal{T}^{\text{high}}_{\mathbf{H}_k(x)[t]} \circ \mathcal{T}^{\text{low}}_{\mathbf{L}_k(x)[t]} \circ \mathcal{T}^{\text{pitch}}_{\mathbf{S}_k(x)[t]} \circ \mathcal{T}^{\text{gain}}_{\mathbf{G}_k(x)[t]}(P_k)\,, \tag{5}$$

where $\mathcal{R}_k(x)[t]$ denotes the $t$-th timestamp of the reconstruction, and $\circ$ denotes the composition of transformations. Note that the chosen transformations are constrained and limited on purpose, as we want each prototype to represent a restricted and consistent set of audio samples.

We measure the quality of the reconstruction $\mathcal{R}_k(x)$ of an input sample $x$ by the $k$-th prototype as the spectro-temporal average of their $\ell_2$ distance:

$$\mathcal{L}_{\text{rec}}(x,k) = \frac{1}{T}\sum_{t=1}^{T} \|x[t] - \mathcal{R}_k(x)[t]\|^2\,. \tag{6}$$

Note that since this function is differentiable and continuous, it can be used as a loss function to train prototypes and transformation networks as defined in the next sections.

### 3.2 Unsupervised Training

We propose an unsupervised learning scheme in which we do not have access to labels $y_1, \cdots, y_N$ during training. Our goal is to cluster the samples $x_1, \cdots, x_N$ into meaningful groups of sounds that can be well approximated by the same prototype spectrogram and its associated transformations. We can train our model by minimizing the following loss:

$$\mathcal{L}_{\text{clustering}} = \sum_{n=1}^{N} \min_{k=1}^{K} \mathcal{L}_{\text{rec}}(x_n, k)\,. \tag{7}$$

This loss is the sum of the reconstruction errors with optimal cluster assignment and is a classical clustering objective used by K-means-based methods.

### 3.3 Supervised Training

In the supervised setting, each prototype and associated transformation networks are dedicated to a class and trained to reconstruct the samples from this class only. Once trained, the model predicts for an input $x$ the class of the prototype with the best reconstruction.

To promote better class discrimination, we propose a dual reconstruction-prediction objective. We associate each input $x$ with a probabilistic class prediction defined as the softmax of the negative reconstruction error between prototypes. For an input $x$ of label $y$, we define the prediction loss $\mathcal{L}_{\text{ce}}$:

$$\mathcal{L}_{\text{ce}}(x,y) = -\log\left(\frac{\exp\left(-\beta\mathcal{L}_{\text{rec}}(x,y)\right)}{\sum_{k=1}^{K}\exp\left(-\beta\mathcal{L}_{\text{rec}}(x,k)\right)}\right)\,, \tag{8}$$

where $\beta$ is a learnable parameter. This loss encourages each prototype to specialize in reconstructing the samples of its associated class. To train our network, we use a weighted sum of both losses:

$$\mathcal{L}_{\text{classif}} = \sum_{n=1}^{N} \mathcal{L}_{\text{rec}}(x_n, y_n) + \lambda_{\text{ce}}\mathcal{L}_{\text{ce}}(x_n, y_n)\,, \tag{9}$$

with $\lambda_{\text{ce}}$ an hyperparameter set to $0.01$.

### 3.4 Parameterization and Training Details

**Architecture.** We implement the functions $\mathbf{G}_k$, $\mathbf{S}_k$, $\mathbf{L}_k$ and $\mathbf{H}_k$ as U-Net style neural networks [23] operating on

| | SOL [6, 7] | | | | | | LibriSpeech [8] | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | w/o supervision | | | w supervision | | | w/o supervision | | | w supervision | | |
| | OA | AA | $\mathcal{L}_{\text{rec}}$ | OA | AA | $\mathcal{L}_{\text{rec}}$ | OA | AA | $\mathcal{L}_{\text{rec}}$ | OA | AA | $\mathcal{L}_{\text{rec}}$ |
| Raw prototypes | 29.3 | 13.2 | 12.4 | 28.4 | 39.5 | 12.9 | 13.3 | 13.2 | 8.6 | 60.5 | 59.5 | 8.6 |
| ⌞ *gain* transformation | 32.2 | 15.8 | 3.4 | 37.4 | 40.1 | 3.8 | 44.4 | 43.3 | 3.4 | 78.7 | 79.1 | 3.4 |
| ⌞ *pitch-shifting* | **37.6** | **19.1** | 2.6 | 51.6 | 46.1 | 2.9 | 46.8 | 46.9 | 2.6 | 77.2 | 77.8 | 2.6 |
| ⌞ *frequency-filters* | 33.4 | 14.6 | **1.5** | 55.1 | 47.7 | **2.1** | **48.8** | **49.5** | **1.6** | 88.8 | 89.4 | **1.6** |
| ⌞ cross-entropy loss | — | — | — | **98.9** | **94.5** | 2.5 | — | — | — | **99.9** | **99.9** | 1.9 |

**Table 1**: **Ablation Study.** We show the impact of the increasing complexity of our modeling on datasets' validation sets.

| | OA | AA |
|---|---|---|
| **SOL** [6, 7] | | |
| Autoencoder + K-means [22] | 28.7 | 12.3 |
| † APNet [17] + K-means [22] | **37.3** | **18.2** |
| Ours w/o supervision | 34.5 | 15.4 |
| **LibriSpeech** [8] | | |
| Autoencoder + K-means [22] | 11.0 | 11.1 |
| † APNet [17] + K-means [22] | 36.3 | 36.4 |
| Ours w/o supervision | **48.6** | **49.5** |

**Table 2**: **Clustering Results.** Clustering performances on the test sets. †: Note that APNet requires labels at training time.

| | OA | AA | $\mathcal{L}_{\text{rec}}$ |
|---|---|---|---|
| **SOL** [6, 7] | | | |
| Direct Classification | 97.8 | 94.8 | — |
| APNet [17] | 95.3 | 91.3 | **0.1** |
| Ours w supervision | **99.3** | **95.8** | 2.6 |
| **LibriSpeech** [8] | | | |
| Direct Classification | 99.4 | 99.5 | — |
| APNet [17] | 97.8 | 97.8 | **0.2** |
| Ours w supervision | **99.9** | **99.9** | 2.6 |

**Table 3**: **Classification Results.** Accuracy and reconstruction error computed on the test sets.

the log-Mel spectrograms $x \in \mathbb{R}^{T \times F}$. To save parameters, all networks share the same encoder, defined as a sequence of 2-dimensional spectro-temporal convolutions and pooling operations. This encoder produces a sequence of feature maps $z_0, \cdots, z_R$ with decreasing temporal and spectral resolutions $T/2^r \times F/2^r$. The spectral dimension of these maps is then collapsed using learned pooling operations: $\tilde{z}_r$ is of resolution $T/2^r$. Each decoder is then defined as a sequence of 1-dimensional temporal convolutions and unpooling operations. The spectral maps $\tilde{z}_0, \cdots, \tilde{z}_R$ are used through skip-connections in the decoders. The prototypes $P_1, \cdots, P_K$ and the inverse temperature $\beta$ are directly trainable parameters of the model.

**Curriculum Learning.** We ensure the stability of our model by gradually increasing its complexity along the training procedure. First, we learn the raw prototypes without any transformation. Once the training loss does not decrease for 10 straight epochs, we add the gain transformation. We then add the pitch shift and spectral filters successively following the same procedure.

**Implementation details.** We use the ADAM [24] optimizer with a learning rate of $10^{-4}$, and a weight decay of $10^{-6}$ for the transformation networks and 0 for $\beta$ and the prototypes. Our model has 545k parameters with $K = 128$ prototypes. For comparison, the reconstruction model proposed by Zinemanas *et al*. [17] has 1.8M parameters.

## 4. EXPERIMENTS

We evaluate our approach in both supervised and unsupervised settings and on two datasets.

**Datasets.** We consider the following datasets:

• **SOL [6,7].** This dataset contains 24 450 samples of individual notes played with various playing techniques by 33 different instruments and sampled at 44.1kHz. We evaluate instrument classification.

• **Librispeech [8].** This 1000-hour corpus contains English speech sampled at 16kHz. We selected the 128 predominant speakers from the `train-clean-360` set. We evaluate the classification of speaker.

For both datasets, we randomly selected 70% of the clips for training, 10% for validation, and 20% for testing. We always use as many prototypes as the number of classes in the dataset.

**Metrics.** To assess the quality of classification models, we report the following metrics:

• *Overall Accuracy (OA).* Percentage of input samples correctly classified by our model, *i.e*., best reconstructed by the prototype assigned to their true class.

• *Average Accuracy (AA).* Average of the classwise accuracy, computed across classes without weights.

• *Reconstruction error ($\mathcal{L}_{rec}$).* To assess the quality of the reconstruction, we also report the reconstruction error $\mathcal{L}_{\text{rec}}$.

Methods trained in the unsupervised setting cluster the

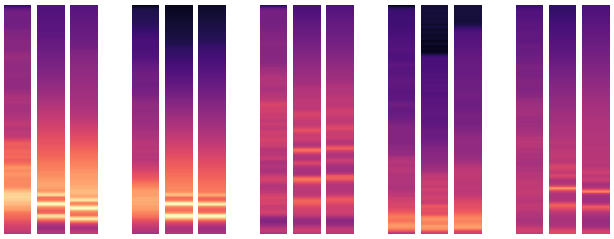



**Figure 4**: **Learned Meaningful Prototypes.** Comparison between the average spectrogram from one class (left), and the prototypes learned in the supervised setting without (middle) and with (right) cross-entropy loss. Our model can learn spectral representations that are crisper and more discriminative than a simple average.

audio samples instead of classifying them. We associate each cluster with the majority class of its assigned training samples. We can then predict for each test sample the class of its best-fitting cluster, allowing us to compute all the classification metrics.

**Baselines.** To put the performance of our method in context, we trained different baselines:

• *Classification.* We trained a temporal convolutional network to classify log-Mel spectrograms. We supervise this network with the cross-entropy on the labels. This network, which does not provide a reconstruction error, is called *Direct Classification*. We also evaluated the APNet [17] approach on our datasets. However, this method is limited in its interpretability, as it relies on latent prototypes and uses a fully connected layer to classify samples based on their similarity to the prototypes.

• *Clustering.* We trained a two-dimensional convolutional autoencoder without skip-connections to reconstruct log-Mel spectrograms. We supervised this network with the $\ell_2$ norm between the input and the reconstruction. We then use K-means [22] on the innermost feature maps to cluster the samples. We also use K-means directly on the predicted similarities between feature maps and APNet prototypes [17]. Note this last approach requires to first train APNet in a supervised setting.

### 4.1 Quantitative Results

**Reconstruction.** As can be seen in Table 1, each successive addition of a transformation decreases the reconstruction error $\mathcal{L}_{\text{rec}}$ and improves the accuracy of classification. This shows that the curriculum training scheme allows our model to learn insightful and complementary spectral transformations. Supervised with the cross-entropy loss, our model reaches high accuracy, at the cost of a slightly higher reconstruction error. In fact, the additional objective not only encourages the reconstruction models $\mathcal{R}_k$ to provide a good reconstruction for the samples of their assigned class, but also a worse reconstruction for other classes.

Finally, we see that while adding frequency-filters yields better reconstruction, they hurt classification performance on SOL [6, 7] in the unsupervised setting. We hypothesize that the filters makes the transformation too expressive, reducing the specificities of the prototypes.

**Clustering.** As shown in Table 2, our method performs well compared to the baselines. For SOL, our method is affected by the class imbalance between samples, but still produces competitive results. In particular, note that our approach does not need any labels during training, in contrast to the APNet-based approach [17]. For LibriSpeech, our method produces convincing clustering results for speaker identification.

**Classification.** As shown in Table 3, our method reaches almost perfect precision for both datasets. While APNet [17] reaches better reconstruction scores, this method learns powerful transformations in an abstract latent space. This leads to a model that is more expressive but less interpretable. In contrast, we restrict the scope of the spectral transformations so that a prototype relates in a meaningful way to the samples that it reconstructs well. Yet, our model is still capable of producing faithful reconstructions, which would be suitable for audio generation tasks.

### 4.2 Interpretability

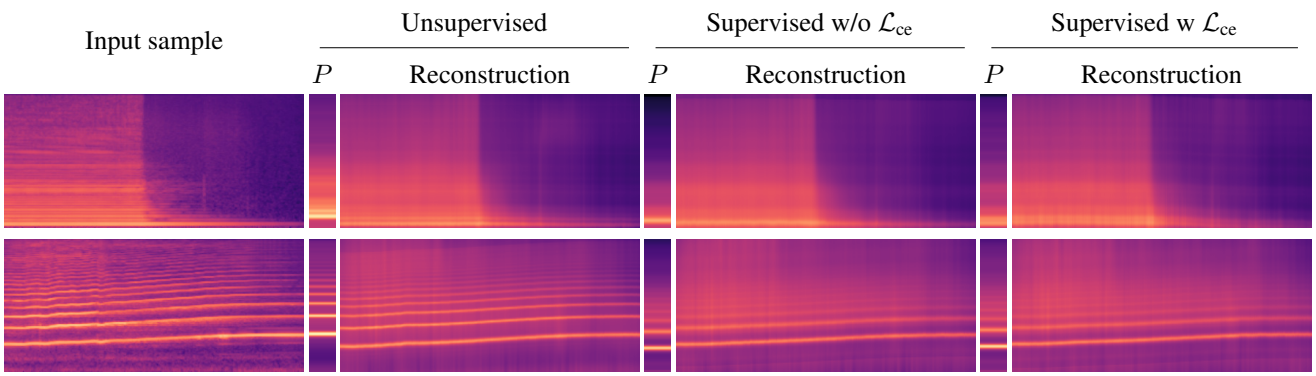
As shown in Figure 5, our model learns to reconstruct complex input samples (*e.g.* different notes, techniques, or voices) by automatically adjusting the amplitude, pitch, and spectral support of spectral prototypes. As seen on Figure 4 and Figure 1, these prototypes capture rich spectral characteristics such as timbre, and can be interpreted as an "acoustical fingerprint". Furthermore, as the prototypes are given in the spectral domain, they can be easily played and analyzed. Audio examples featuring the learned prototypes are provided in the supplementary materials.

In Figure 6, we represent the reconstruction of sounds played by various instrument with different prototypes. While the reconstructions are clearly more faithful when using the instruments' dedicated prototype, this opens the perspective for further MIR application such as interpretable timbre transfer.

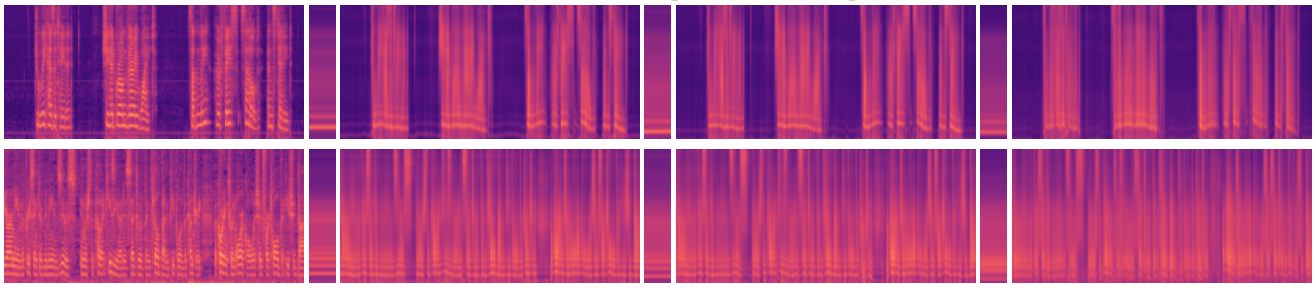
## 5. CONCLUSION

We presented a novel approach to audio understanding by representing large collections of audio clips with few spectral prototypes equipped with learned transformation networks. Our model produces concise, expressive, and interpretable representations of raw audio excerpts that can be used to obtain state-of-the-art results for audio classification and clustering tasks.

**Acknowledgements.** This work was supported in part by ANR project READY3D ANR-19-CE23-0007 and was granted access to the HPC resources of IDRIS under the allocation 2022-AD011012096R1 made by GENCI. We thank Theo Deprelle, Nicolas Gonthier, Tom Monnier and Yannis Siglidis for inspiring discussions and feedbacks.




(a) **SOL [6, 7].** Reconstructions of an audio clip of contrabass (top) and clarinet (bottom).



(b) **LibriSpeech [8].** Reconstructions of an audio clip of C. Nendza (F) (top) and C. Berriux (M) (bottom) speaking.

**Figure 5**: **Reconstruction of Input Sounds.** For each input sample, we show the reconstruction provided by different settings of our model as well as the corresponding prototype $P$ used for reconstruction. Note how the timbre of the speaker or instrument is well represented while the inputs fluctuates in pitch and intensity. This leads to an insightful characterization of each instrument or speaker.

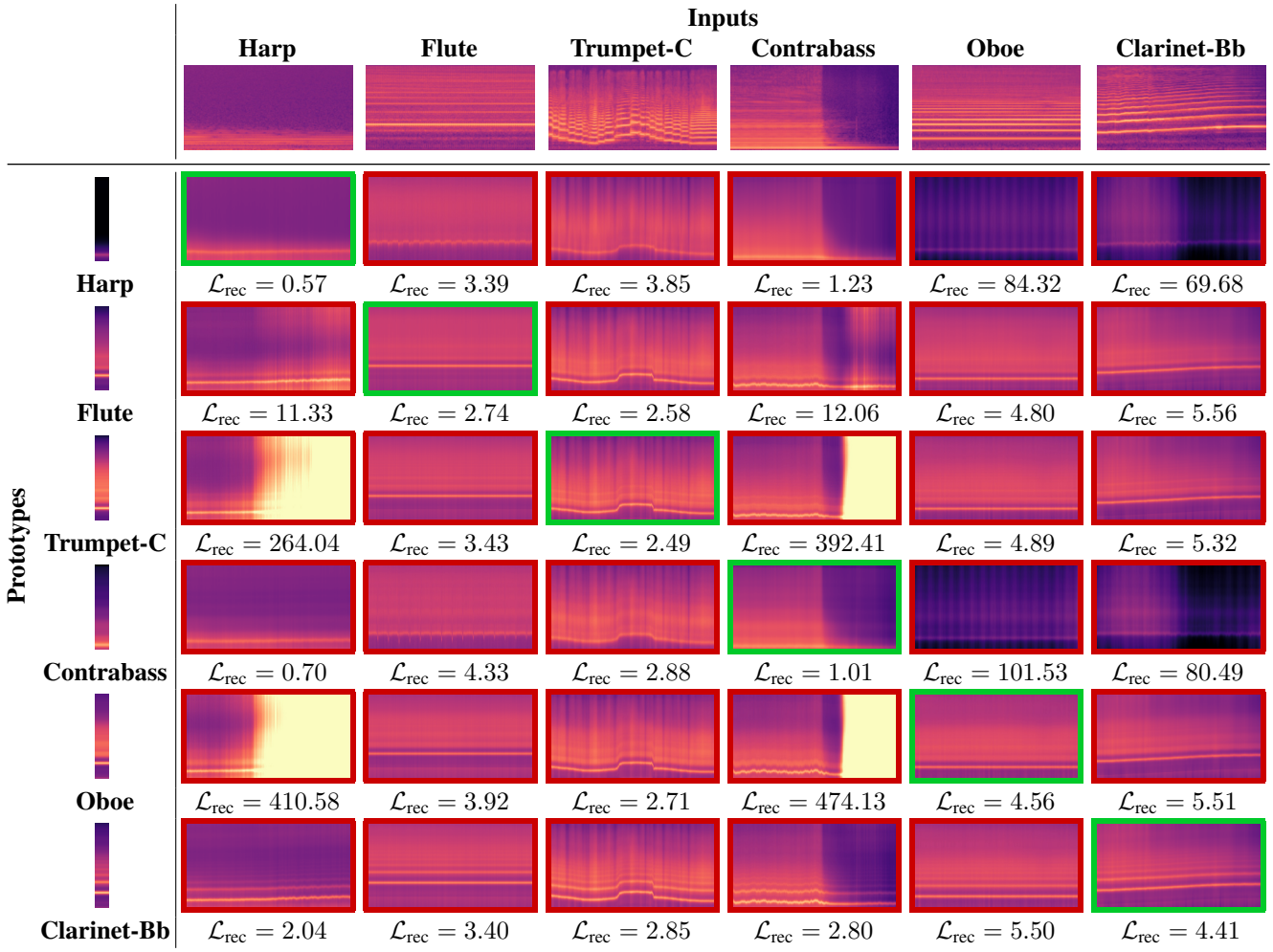



**Figure 6**: **Reconstructions from Different Prototypes.** We show the reconstructions of input samples (columns) from SOL [6, 7] by learned prototypes (lines) in the supervised setting without cross-entropy loss. The prototype of the corresponding instruments provides the lowest reconstruction error (green frame).

## 6. REFERENCES


[1] J. Abeßer, "A review of deep learning based methods for acoustic scene classification," *Applied Sciences*, 2020.

[2] N. Ndou, R. Ajoodha, and A. Jadhav, "Music genre classification: A review of deep-learning and traditional machine-learning approaches," in *IEMTRONICS*, 2021.

[3] T. Monnier, T. Groueix, and M. Aubry, "Deep Transformation-Invariant Clustering," in *NeurIPS*, 2020.

[4] R. Loiseau, T. Monnier, M. Aubry, and L. Landrieu, "Representing Shape Collections with Alignment-Aware Linear Models," in *3DV*, 2021.

[5] M. Jaderberg, K. Simonyan, and A. Zisserman, "Spatial transformer networks," *NeurIPS*, 2015.

[6] G. Ballet, R. Borghesi, P. Hoffmann, and F. Lévy, "Studio online 3.0: An internet "killer application" for remote access to IRCAM sounds and processing tools," in *Journées d'Informatique Musicale*, 1999.

[7] C. E. Cella, D. Ghisi, V. Lostanlen, F. Lévy, J. Fineberg, and Y. Maresz, "OrchideaSOL: a dataset of extended instrumental techniques for computer-aided orchestration," *ICMC*, 2020.

[8] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *ICASSP*, 2015.

[9] D. Bhalke, C. Rao, and D. S. Bormane, "Automatic musical instrument classification using fractional fourier transform based-MFCC features and counter propagation neural network," *Journal of Intelligent Information Systems*, 2016.

[10] V. Lostanlen, J. Andén, and M. Lagrange, "Extended playing techniques: the next milestone in musical instrument recognition," in *International Conference on Digital Libraries for Musicology*, 2018.

[11] Z. Bai and X.-L. Zhang, "Speaker recognition based on deep learning: An overview," *Neural Networks*, 2021.

[12] F. Ye and J. Yang, "A deep neural network model for speaker identification," *Applied Sciences*, 2021.

[13] N. Tits, F. Wang, K. E. Haddad, V. Pagel, and T. Dutoit, "Visualization and interpretation of latent spaces for controlling expressive speech synthesis through audio analysis," *Conference of the International Speech Communication Association*, 2019.

[14] F. Roche, T. Hueber, S. Limier, and L. Girin, "Autoencoders for music sound modeling: a comparison of linear, shallow, deep, recurrent and variational models," *Sound and Music Computing*, 2018.

[15] J. Naranjo-Alcazar, S. Perez-Castanos, P. Zuccarello, F. Antonacci, and M. Cobos, "Open set audio classification using autoencoders trained on few data," *Sensors*, 2020.

[16] O. Li, H. Liu, C. Chen, and C. Rudin, "Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions," in *AAAI*, 2018.

[17] P. Zinemanas, M. Rocamora, M. Miron, F. Font, and X. Serra, "An interpretable deep learning model for automatic sound classification," *Electronics*, 2021.

[18] T. Monnier, E. Vincent, J. Ponce, and M. Aubry, "Unsupervised Layered Image Decomposition into Object Prototypes," in *ICCV*, 2021.

[19] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *The Journal of the Acoustical Society of America*, 1937.

[20] H. Tamaru, Y. Saito, S. Takamichi, T. Koriyama, and H. Saruwatari, "Generative moment matching network-based random modulation post-filter for dnn-based singing voice synthesis and neural double-tracking," in *ICASSP*, 2019.

[21] M. Andreux and S. Mallat, "Music generation and transformation with moment matching-scattering inverse networks," in *ISMIR*, 2018.

[22] L. Bottou and Y. Bengio, "Convergence properties of the k-means algorithms," in *NeurIPS*, 1995.

[23] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *MICCAI*, 2015.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.

# AN EXPLORATION OF GENERATING SHEET MUSIC IMAGES

**Marcos Acosta**
Harvey Mudd College
mdacosta@g.hmc.edu

**Irmak Bukey**
Pomona College
ibab2018@mymail.pomona.edu

**TJ Tsai**
Harvey Mudd College
ttsai@g.hmc.edu

## ABSTRACT

Many previous works in recent years have explored various forms of music generation. These works have focused on generating either raw audio waveforms or symbolic music. In this work, we explore the feasibility of generating sheet music images, which is often the primary form in which musical compositions are notated for other musicians. Using the PrIMuS dataset as a testbed, we explore five different sequence-based approaches for generating lines of sheet music: generating sequences of (a) pixel columns, (b) image patches, (c) visual word tokens, (d) semantic tokens, and (e) XML-based tags. We show sample generated images, discuss the practical challenges and problems with each approach, and give our recommendation on the most promising paths to explore in the future.

## 1. INTRODUCTION

This work explores the feasibility of generating music in the form of sheet music images. Below, we provide a brief overview of recent work in the area of music generation, motivate the problem of generating sheet music images, and describe five sequence-based approaches that we will explore in this study. Our main goal is to identify the practical challenges and problems with each of the five approaches and to provide a recommendation on the most promising paths to explore in the future.

Music generation has been an active research topic of interest to the MIR community in recent years. Many recent works in this area fall into one or more of the following three research thrusts. The first research thrust is controllable music generation, in which a user can control certain aspects of the music generation process. Some examples of this include providing a template from which variations can be generated [1] [2], providing a video to which a suitable background music track is generated [3], specifying lyrics for the generated song [4], or disentangling different characteristics of music like style & melody [5], pitch & rhythm [6] [7], or structure & content [8]. A second research thrust is generating music with realistic short-term and long-term structure. Most recent works achieve this by representing music as a sequence of discrete tokens and using Transformer-based architectures to learn long-term dependencies. Some prominent examples of this include OpenAI's Jukebox model [4] and Google Magenta's Music Transformer model [9] [10]. Other works focus on encoding music in a way that allows the generated music to exhibit appropriate metrical structure at the beat, bar, and phrase levels [11]. A third research thrust is to generate music in different contexts and musical representations. Some examples include multi-part music [12], multi-track music [13] [14] [15] [16], guitar tabs [17], and jazz lead sheets [18]. Much of this work focuses on how to encode different music representations or different aspects of music (like pitch, duration, velocity, and timing [11]) in a form that is suitable for training with a Transformer model.

This work falls into the last group and explores music generation in a different musical representation: sheet music images. To the best of our knowledge, this is the first systematic study of generating sheet music images.

Why generate music in the form of sheet music images? We present four reasons. First, sheet music is the predominant medium by which compositions are shared in many genres of music. If a piece of music is intended to be a composition (as opposed to a demo for an academic research paper), the expectation is that the composition will be in the form of sheet music. Second, sheet music contains important musical information that is not contained in MIDI files (which is the most common symbolic music representation). For example, musical symbols in the sheet music like bar lines, rests, note ties, phrasing markings, sharps and flats, and key changes may not appear in a generated MIDI file in any explicit way. Third, generating sheet music cleanly decouples the musical composition and expressive performance aspects of music, allowing us to focus exclusively on the composition problem. Fourth, there is an abundance of sheet music data (>650k scores) available to the research community through the International Music Score Library Project (IMSLP) [19]. For the above reasons, we believe that studying the generation of sheet music images is an interesting and relevant problem.

Our goal is to explore the feasibility of generating sheet music. As an initial step, we focus on the specific task of generating monophonic incipit images, which are short snippets of sheet music that are used to identify a musical work. Using the PrIMuS dataset [20] as a testbed, we explore five different sequence-based approaches for generating incipits: (a) generating sequences of pixel columns, (b) generating sequences of image patches, (c) generating sequences of visual word tokens, (d) generating sequences

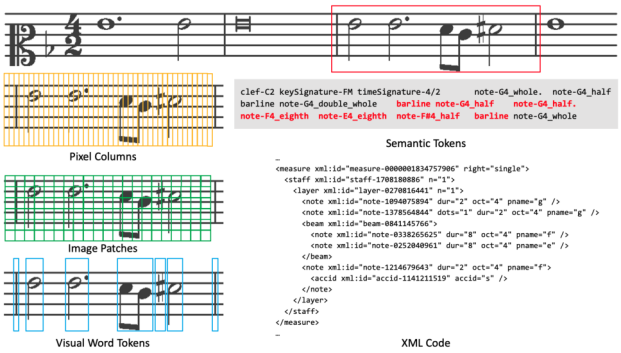



**Figure 1**. An example incipit from the PrIMuS dataset (top). We explore five different ways to generate sheet music: generating pixel columns, image patches, visual word tokens, semantic tokens, and XML code.

of semantic tokens, and (e) generating MEI code, an XML-based sheet music representation. For each of these five approaches, we train GPT-2 [21] and AWD-LSTM [22] language models, generate example incipits, and characterize the main challenges and problems with each approach.

## 2. SYSTEM DESCRIPTIONS

In this section we describe five different sequence-based approaches for generating a monophonic incipit image. All five approaches use language models to generate sequences of tokens, but they differ in the way that the tokens are constructed or defined.

### 2.1 Generating Pixel Columns

The first approach is to generate sequences of pixel columns. This is done in five steps, which are described in the following five paragraphs.

The first step is to standardize the incipits. The raw incipits in the PrIMuS dataset have variable height and width, so it is necessary to standardize the height of the incipit. This is done by selecting a fixed height for all incipits, ensuring that the five staff lines are vertically centered in the resulting image. This ensures that the staff lines appear in the same vertical pixel positions in each image. After the standardization, the images have a fixed height but variable width.

The second step is to convert pixel columns to words. Here, we interpret each standardized image as a sequence of words, where each word corresponds to a single pixel column. We convert pixel columns to words by simply binarizing the (grayscale) pixel values and interpreting each column's binary representation as a word.

The third step is to train a language model on the word sequences. This can be done in a self-supervised manner by predicting the next word in a sequence. We ran experiments with two different language models: (a) the AWD-LSTM model [22], which is a 3-layer LSTM model that incorporates multiple forms of dropout and (b) the GPT-2 small model [21], which is a 6-layer Transformer decoder model. We use the implementation of AWD-LSTM from the fastai [23] library and the implementation of GPT-2 from the huggingface [24] library. We use the default recommended vocabulary size in each respective library (GPT-2 small 30k, AWD-LSTM 60k), and map infrequently occurring words to a special unknown word token <unk>.

The fourth step is to generate new word sequences given a starting seed sequence. We do this by generating a new token at each time step and using the generated token as input to the next time step. For AWD-LSTM, we sample from the softmax distribution with a temperature of 0.8 (default in fastai). For GPT-2, we use top $K = 50$ sampling (default in huggingface).

The fifth step is to render the generated word sequence as an image. Since each word is a binary string of 0's and 1's corresponding to a single pixel column, we simply concatenate the generated words in a matrix and directly render it as a black-and-white image.

### 2.2 Generating Image Patches

The second approach is to generate sequences of image patches. This is done in five steps, which are described in the following five paragraphs.

The first step is to standardize the incipits. This is done in the same manner as described in Section 2.1 above.

The second step is to convert image patches to words. Figure 1 shows a graphical illustration of this process for a single measure (green squares at left). Instead of using pixel columns as words, we can instead use $15 \times 15$ square image patches as words. This allows individual words to retain context in both the vertical and horizontal dimensions, rather than only along the vertical dimension. This approach has been shown to work effectively for image classification in the Vision Transformer model [25]. After breaking the incipit image into square patches, we form a sequence by considering patches from top to bottom and from left to right. Each image patch is binarized and interpreted as a single word.

The third step is to train a language model on the word sequences. This is done in the same manner as before. We tried training both with and without special <col> tokens indicating the end of each column of patches but found that including the <col> tokens did not help.

The fourth step is to generate new word sequences given a starting seed sequence. We generate a sequence of words autoregressively as described previously.

The fifth step is to render the generated word sequence as an image. Each word is a binary string of length $15 \times 15 = 225$ and can be directly reshaped into a binary image patch. The image patches are assembled in the same rasterized order to generate the incipit image.

### 2.3 Generating Visual Word Tokens

The third approach is to generate sequences of visual word tokens. This is done in five steps, which are described in the following five paragraphs.

The first step is to standardize the incipits. This is done in the same manner as described in Section 2.1.