GrooveGen: Input-Conditioned Drum Loop Generator

Authors: Avinash Duggal and Ethan Kusnadi

Date: December 2, 2025

Abstract

This report presents GrooveGen, a generative model designed to create realistic, stylistically controlled drum grooves based on user-defined stylistic inputs such as "funk 100 BPM". Training on the Google Groove MIDI Dataset, we develop a Conditional Variational Autoencoder (CVAE) that learns a latent space of expressive drum patterns and generates new sequences conditioned on an input tag vector (18 genres + normalized BPM). The CVAE produced genre-consistent drum patterns, with the latent space achieving 51.26% SVM classification accuracy and generated examples reaching 56.32% accuracy under a CNN genre classifier, as well as positive qualitative listening tests. These results show that conditioning on genre and tempo enables meaningful control over generated grooves, possibly leading to more advanced input-driven music tools.

1. Introduction

As generative AI becomes increasingly important to the world, music generation has naturally emerged as an interesting field. Many musicians can play instruments like guitar or piano but lack drumming skills or access to a drummer. While they could program drum patterns manually, this is time-consuming and requires rhythmic expertise. AI-generated tools could address this

need, but current approaches are still limited. Beyond legal questions surrounding authorship and ownership, AI-generated music is often rhythmically incoherent or musically uninteresting. Generating humanlike and genre-consistent drum patterns remains a challenge.

In this project, we develop GrooveGen, a generative model that produces realistic drum patterns conditioned on user-specified stylistic tags. To achieve this, we implement a Conditional Variational Autoencoder (CVAE) and train it on the Groove MIDI Dataset developed by Google, which contains over 22,000 measures of expressive, human-played drumming [1]. We evaluate the model using an SVM to judge the latent space, a CNN genre classifier, and qualitative human listening tests.

2. Background

2.1 Dataset

We utilized the Groove MIDI Dataset (GMD) provided by Google [1]. This dataset consists of 13.6 hours, 1150 MIDI files, and over 22,000 measures of expressive human-played drumming. The drum performances span 18 genres: afrobeat, afrocuban, blues, country, dance, funk, gospel, highlife, hiphop, jazz, latin, middle eastern, new orleans, pop, punk, reggae, rock, and soul. The key features of the dataset for our purposes are drum kit component usage, hit offsets, hit velocity, and beats per minute (BPM).

2.2 Related Work

In the dataset introductory paper, the authors' GrooVAE model uses a variation of a sequence-to-sequence model which encodes drum score inputs into a single latent vector using a

bidirectional LSTM and decodes them into full expressive drum performances using a 2-layer LSTM [1]. The model explicitly predicts hit timing offsets and velocities, allowing it to generate patterns that sound closer to real drum performances. GrooVAE focuses on humanization and performance modeling, while our project focuses on conditional generation based on the genre and tempo.

3. Data Processing Pipeline

3.1 Filtering and Parsing

Because the Groove MIDI Dataset was released by Google with built-in TensorFlow Datasets support, we were able to load it directly in TensorFlow using the authors' predefined train/validation/test splits. These splits are defined at the performance level, which prevents the model from memorizing an individual drummer's style and makes sure that all segments from the same performance remain in the same split. We chose to use the 4bar-midionly configuration of the dataset, which segments each performance into 4-bar sequences and excludes audio files.

We filtered for 4/4 performances for simplicity. We also mapped raw MIDI pitches to 9 canonical drum classes (kick, snare, 3 toms, 2 hi-hats, ride, crash), following the reduced drum mapping introduced in [2] and adopted in the GrooveVAE work in [1]. Finally, we quantized all sequences to a 16th-note grid, as done in prior studies to match what is commonly found in music software [1], [3].

3.2 MIDI to HOV Representation

Each 4-bar window (64 sixteenth notes × 9 instruments × 3 channels) is decomposed into three

separate matrices following the representation used in previous work [1]:

- H (Hits): A binary 64 × 9 matrix of onsets (drum hits) indicating whether each kit component is struck at each timestep.
- O (Offsets): A 64 × 9 matrix of microtiming deviations in [-0.5, 0.5) showing how far each hit falls from the quantized 16th-note grid.
- V (Velocities): A 64 × 9 matrix of normalized velocities in [0, 1] to quantify how hard each hit was.

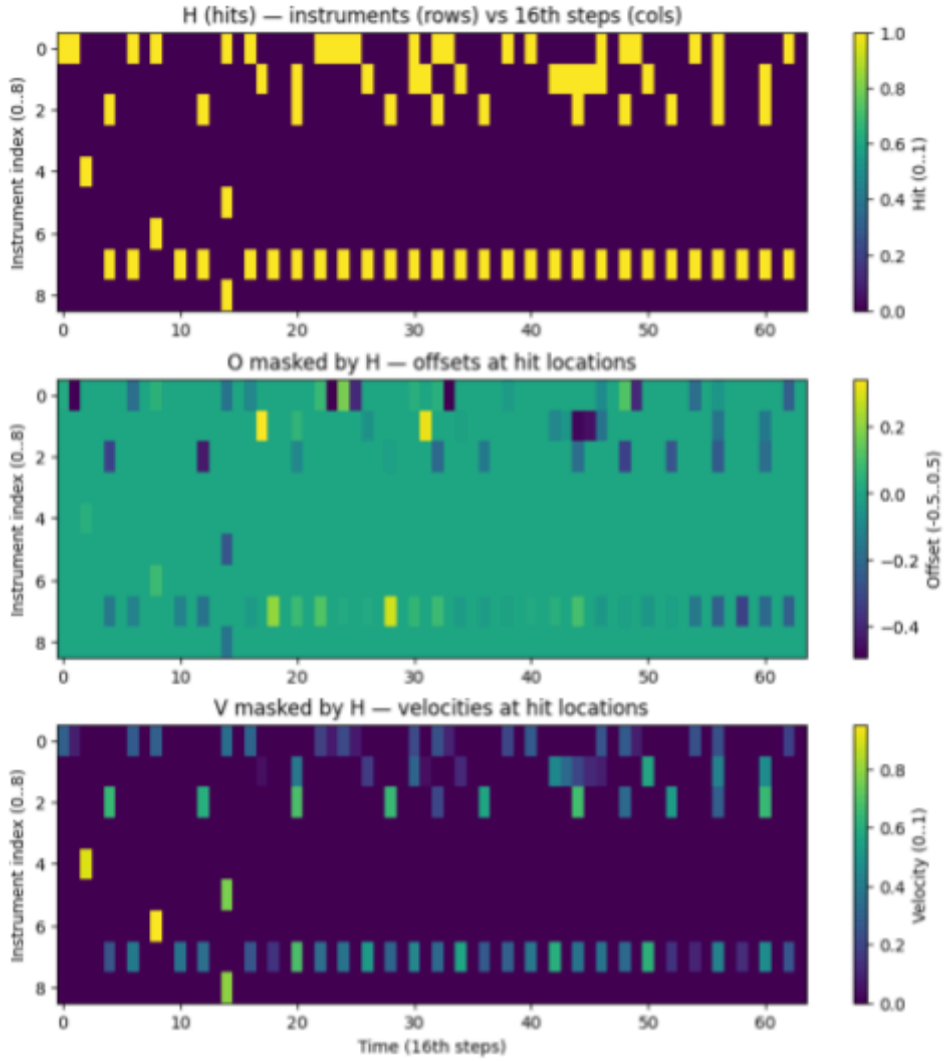

Figure 1. Example 4-bar window represented in the HOV format.

To prepare the dataset for input for our CVAE, we converted each MIDI sequence into its Hit–Offset–Velocity (HOV) representation, then constructed a 19-dimensional conditioning vector which combines a one-hot encoding of the musical styles and a normalized BPM value.

4. Model Architecture

4.1 Conditional VAE Overview

The core of our generation engine relies on a CVAE, which allows for sampling new drum loops conditioned on a 19-dimensional vector (18 genres + normalized BPM). We chose a Conditional Variational Autoencoder (CVAE) as our generative model for several key reasons. The first reason was probabilistic generation. Unlike deterministic models, VAEs learn a probability distribution over the latent space, enabling diverse outputs from the same conditioning inputs. This is crucial for creative music generation, where we want variation rather than deterministic repetition. The second reason was the structured latent space. VAEs enforce a continuous, smooth latent space, which makes it possible to interpolate between different drum styles and create new combinations. By extending the standard VAE to a CVAE, we can explicitly control generation through our 19-dimensional conditioning vector, allowing users to specify their desired output characteristics.

4.2 Encoder

The encoder maps input patterns (64 × 9 × 3) and conditioning vector (19-dim) to a 128-dimensional latent space:

- Flatten the 9 instruments × 3 features into 27 channels to process the temporal dimension with 1D convolutions. This preserves the sequential nature of drum patterns while treating

all instrument features as parallel channels.

- Use convolutional downsampling on four Conv1D layers (64→128→256→512 channels, kernel=4, stride=2) with BatchNorm and ReLU, progressively compressing 64 steps to 4 steps, forcing the model to learn hierarchical representations with low-level (individual hits), mid-level (beat patterns), and high-level (bar-level groove structure)
- Drum patterns are inherently sequential. 1D convolutions along the time axis capture recurring patterns more effectively than treating time and instruments as two spatial dimensions. The kernel size of 4 covers one beat (4 sixteenth notes), allowing the model to learn beat-level patterns.
- Obtain the latent mean and log variance using dense layers of 512 and 256
- Outputs latent mean and log variance (128-dim), and are sampled via the reparameterization trick. We experimented with 64, 128, and 256 dimensions, finding 128 provided the best reconstruction quality while maintaining diversity in generated samples.

4.3 Decoder

The decoder reconstructs drum patterns from latent vector z (128-dim) and conditioning vector c (19-dim):

- Similar to the encoder, we inject conditioning early in the decoder to guide reconstruction throughout all upsampling layers.
- Transposed convolutions perform learned upsampling, progressively expanding the compressed latent representation back to full 64-step resolution. This mirrors the encoder's downsampling path and allows the model to generate coherent patterns at multiple time scales.

- Perform learned upsampling on 4 Conv1DTranspose layers (256→128→64→32 channels) with BatchNorm, expanding 4 steps to 64 steps
- Output reshaped back to original dimensionality (64 × 9 × 3), which represents the Hits, Offsets, and Velocities
- Channel-Specific Activations:
    - Hits (H): sigmoid → Binary probabilities [0, 1]
    - Offsets (O): tanh → scale by 0.5 → Continuous values [-0.5, 0.5]
    - Velocities (V): sigmoid → Normalized intensities [0, 1]
- Each channel has different semantics and value ranges. Sigmoid naturally models binary hit probabilities. Tanh provides symmetric output for timing deviations (early vs. late hits). Treating all channels identically would force the model to learn these constraints implicitly, making training harder.

4.4 CVAE Loss

Our training objective combines reconstruction loss and KL divergence:

$$L_{total} = L_{recon} + \beta * L_{KL}$$

Reconstruction Loss:

$$L_{recon} = L_{hits} + 0.5 * L_{offset} + 0.5 * L_{velocity}$$

Hit Loss (Binary Cross-Entropy):

$$L_{hits} = -\text{mean}[H_{true} * \log(H_{pred}) + (1 - H_{true}) * \log(1 - H_{pred})]$$

Binary Cross-Entropy is appropriate for binary classification (hit or no hit). It heavily penalizes

confident wrong predictions.

Offset Loss (Masked MSE):

$$L_{offset} = \text{sum}[(O_{true} - O_{pred})^2 * H_{true}] / \text{sum}[H_{true}]$$

We only compute offset loss where hits occur (masked by H_true). This prevents the model from learning meaningless offset values for silent timesteps. MSE is used because offset is a continuous regression target.

Velocity Loss (Masked Mean Squared Error):

$$L_{velocity} = \text{sum}[(V_{true} - V_{pred})^2 * H_{true}] / \text{sum}[H_{true}]$$

Similarly, velocity is only meaningful where hits occur. MSE measures how well the model predicts hit intensity. Hit timing is the most perceptually important aspect of drum patterns. Empirically, we found that equal weighting caused the model to overfit offset/velocity at the expense of hit accuracy. The 1:0.5:0.5 ratio prioritizes getting the basic rhythm right before refining expressive details.

KL Divergence:

$$L_{KL} = -0.5 * \text{mean}[1 + z_{log\_var} - z_{mean}^2 - \exp(z_{log\_var})]$$

The KL term regularizes the latent space to be close to a standard normal distribution N(0, I). This ensures:

- Smooth interpolation: Similar latent codes produce similar outputs
- Sampling capability: We can generate new patterns by sampling $z \sim N(0, I)$
- No posterior collapse: Forces the encoder to use the latent space rather than ignoring it

Standard VAEs (β=1) often suffer from posterior collapse—the decoder learns to ignore z and rely only on conditioning. With β=0.01, we prioritize reconstruction quality while still maintaining a structured latent space, downweighting the KL term. We experimented with $\beta \in \{0.001, 0.01, 0.05, 0.1\}$ and found 0.01 gave the best balance between reconstruction fidelity and generation diversity.

4.5 Training

We implemented the CVAE with TensorFlow and Keras and trained it in Google Colab.

Optimizer: Adam (learning rate = 1e-4)

- Adam adapts learning rates per parameter, crucial for training VAEs where different components (encoder/decoder) may converge at different rates.

Batch Size: 32

- Limited by GPU memory. Larger batches (64+) would improve gradient estimates but risk overfitting on our dataset size.

Regularization:

- Dropout (0.2, 0.5) in dense layers prevents overfitting
- Batch normalization improves training stability
- KL divergence acts as implicit regularization on the latent space

Callbacks:

- EarlyStopping (patience=15): Monitors validation loss and stops if no improvement for 15 epochs
- ReduceLROnPlateau (patience=5, factor=0.5): Reduces learning rate when validation

plateaus, allowing finer optimization

- ModelCheckpoint: Saves best model based on validation loss

5. Evaluation

5.1 SVM Evaluation on CVAE Latent Space

To evaluate whether the CVAE learned a style-structured latent space, we trained a Support Vector Machine (SVM) on the latent space of our encoder. We used the LinearSVC package from scikit-learn, which uses a One-vs-Rest (OvR) strategy for multiclass classification. This allows us to test whether the 18 musical styles in the Groove MIDI Dataset are roughly linearly separable in the learned latent space, providing a diagnostic of how well the CVAE captures stylistic information. The CVAE encoder produces z-mean embeddings, which represent the mean of the posterior distribution and serve as deterministic latent representations for each sample. We trained the SVM on these embeddings and evaluated this classifier on the validation set, achieving an accuracy of 51.26%. Since this accuracy measures how well the 18 genres can be linearly separated in the CVAE's latent space, this result indicates that the model encodes meaningful stylistic patterns despite our limited training resources.

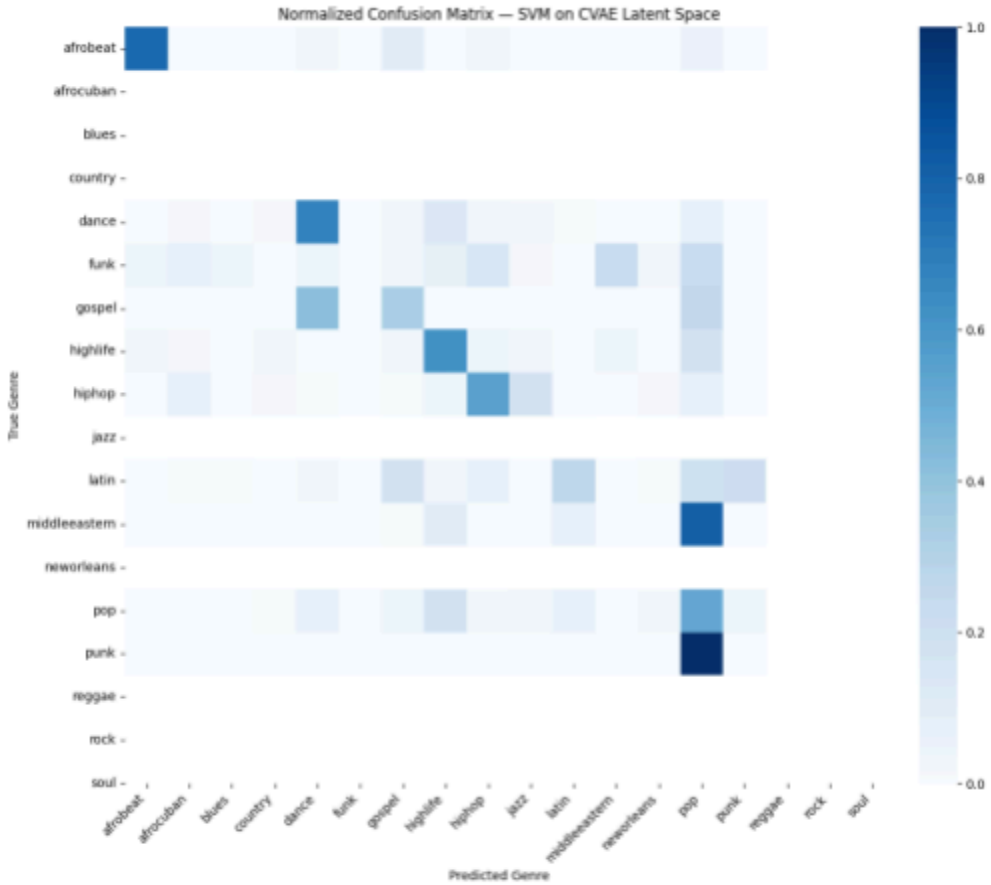

Figure 2. Normalized confusion matrix for the SVM trained on the CVAE latent space.

The confusion matrix generated by the SVM reveals that certain genres such as afrobeat, dance, and highlife form clear clusters in the latent space, while others such as middle-eastern and punk exhibit high confusion with other styles. Several classes are not predicted at all, likely due to class imbalance or poor latent-space separation.

5.2 CNN Classifier for Generated Samples

Additionally, we constructed a Convolutional Neural Network (CNN) with TensorFlow and Keras to evaluate if CVAE-generated samples matched the genre that they were conditioned on. Because the original dataset has class imbalance, we augmented the minority classes using

random velocity scaling, timing jitter, and instrument dropout. After augmentation, we concatenated the H, O, and V matrices along the feature dimension into a single input tensor and passed this directly to the CNN.

Our CNN consists of four convolutional layers with 3x3 kernels, each followed by batch normalization, max-pooling over the time dimension only, and dropout. The filter size increases from 32 to 64 to 128 to 256 to capture progressively higher-level features. After a global average pooling layer and two dense layers, the network feeds to the output layer, an 18-way softmax that predicts the style label. We trained the network with the Adam optimization algorithm at a learning rate of 10^{-3} using sparse categorical cross-entropy loss for 50 epochs.

The CNN achieved a 56.32% accuracy and 5.7064 loss on the test set. During training, the model exhibited significant overfitting and achieved training accuracies as high as 95% while validation accuracy plateaued around 58%. This is shown in Figure 3. The training loss decreased to approximately 0.05, while validation loss remained high at 6-7, indicating the model memorized training patterns rather than learning generalizable features. We employed several regularization strategies to combat this: dropout layers (0.25, 0.5), batch normalization, data augmentation for minority classes, and early stopping with a patience of 10 epochs. The learning rate was reduced on a plateau with a factor of 0.5, ultimately decreasing from 10^{-3} to 6.25×10^{-5} by epoch 48 when early stopping triggered. Despite these efforts, the gap between training and validation performance persisted, likely due to the substantial class imbalance (rock: 5,161 samples vs. highlife: 45 samples) and limited dataset size after the 4/4 time signature filter.

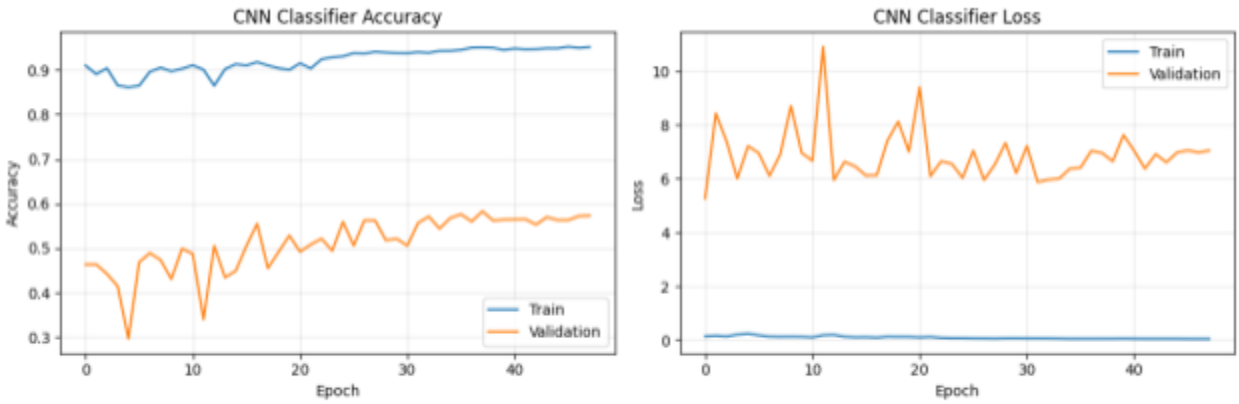

Figure 3: CNN Classifier Accuracy and Loss

To better understand the classifier's performance across genres, we computed ROC-AUC scores using a one-vs-rest approach. The micro-average AUC of 0.8775 and macro-average AUC of 0.8594 indicate strong overall discriminative ability despite the poor accuracy. This discrepancy arises because accuracy treats all classes equally, while AUC measures the model's ability to rank positive examples higher than negative ones, as this is a more nuanced metric for imbalanced datasets.

Per-class AUC scores reveal significant variance across genres. The model excels at identifying punk (0.9986), afrobeat (0.9940), and Latin (0.9898), suggesting these genres have highly distinctive rhythmic patterns. Conversely, reggae (0.5422) and pop (0.6785) prove more challenging, likely due to their rhythmic overlap with other styles and higher within-genre diversity. Jazz (0.8065) and rock (0.8245) fall in the middle range, indicating moderate separability. These results validate that the CNN effectively learns genre-specific patterns in the HOV representation, making it suitable for evaluating whether our CVAE generates stylistically consistent drum loops.

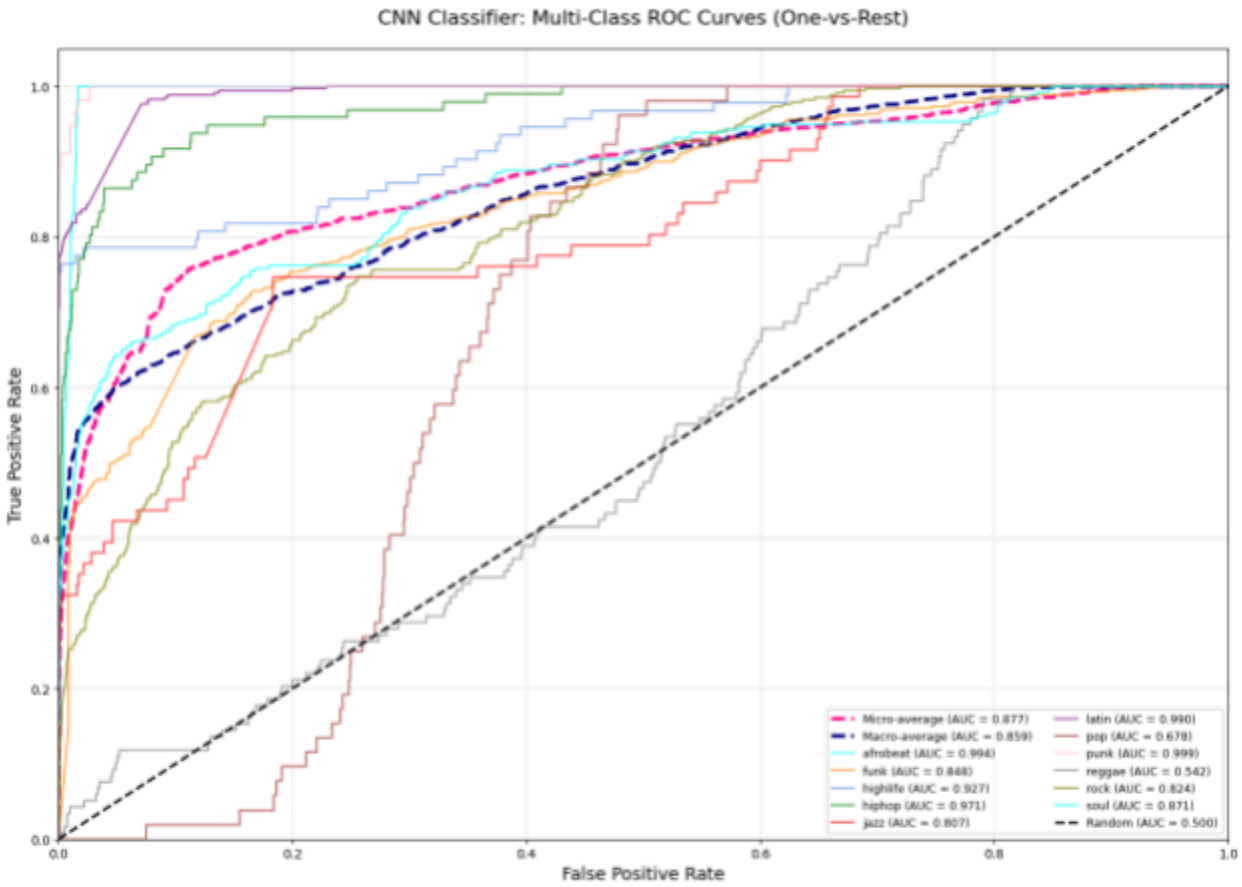

Figure 4. Multi-class ROC curves for the CNN genre classifier

5.3 Qualitative Listening Review

Generated drum loops varied considerably in quality. Some patterns sounded reasonably close to a human performance, while others produced strange rhythms or accents in the wrong places. Across examples, genres were generally distinct, and samples within the same genre often had similar structures.

6. Conclusion

We successfully developed an input-conditioned drum groove generator capable of producing 4-bar sequences. Our Conditional VAE approach demonstrates that conditioning on inputs such

as genre and BPM can effectively guide the generation of realistic drum patterns. Latent space structure (51% SVM accuracy), genre classifier evaluation (56% CNN accuracy), and listening tests together indicate that the CVAE learns meaningful stylistic variation.

Future extensions of this project could include training the CVAE longer and on more data, such as the Expanded Groove MIDI Dataset (EGMD), which contains 444 hours of drum samples. Other models such as diffusion models or Transformer-based architectures could be explored as well. Additionally, we could add more control options to emphasize specific drum kit components. Finally, we could implement natural-language text input so the model can interpret free-form prompts rather than fixed tags.

References

[1] J. Gillick, A. Roberts, J. Engel, D. Eck, and D. Bamman, "Learning to Groove with Inverse Sequence Transformations," in *International Conference on Machine Learning (ICML)*, 2019.

[2] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A hierarchical latent vector model for learning longterm structure in music," arXiv preprint arXiv:1803.05428, 2018.

[3] M. Wherry, “All about quantise,” *Sound on Sound*, 2006.