Fine-Tuning MusicGen for Composer Style Transfer: Leveraging Style Embeddings for Conditioned Music Generation

Stanford CS229 Project

Sara Kothari
Department of Computer Science
Stanford University
sarako@stanford.edu

Yanny Gao
Department of Computer Science
Stanford University
rgao1218@stanford.edu

1 Introduction

Generating music that convincingly mimics the style of a specific composer presents a unique challenge in machine learning. While transformer-based models like MusicGen (Copet et al.) can generate coherent music, they often struggle to capture the subtle stylistic traits that define individual classical composers. Achieving accurate composer style transfer was particularly interesting to the both of us, as we are a team composed of a classical pianist and classical vocalist.

Our primary goal is to enable style transfer based on a prompt such as "Convert this music to Chopin's style". For instance, given a one-minute audio recording of Bach's music as input, our fine-tuned model aims to generate a new piece that reflects Chopin's musical characteristics.

In our project, we achieved three key advancements:

- **Style Encoder:** We trained a style encoder using contrastive loss to produce embeddings that cluster audio samples from the same composer.
- **Fine-Tuning:** We fine-tuned MusicGen (Copet et al.) using parameter-efficient fine-tuning ((PEFT)) to integrate style embeddings and align generated audio with a target composer's style by minimizing the mean squared error (MSE) between the generated audio's embedding and the target composer's mean embedding.
- **Composer Classifier:** We developed a classifier using transfer learning with the MusicNet (Thickstun et al., 2017) dataset that predicts the composer of a one-minute audio sample with **92% validation accuracy**, which we used to evaluate the generated music's alignment with the intended style.

To further evaluate our model's consistency, we employed a **cycle consistency evaluation** that reconstructs music through **Composer A** $\rightarrow$ **Composer B** $\rightarrow$ **Composer A** and measures similarity using **Frechet Audio Distance (FAD)**(Kilgour et al., 2019).

2 Related Work

Deep Generative Models for Audio Synthesis

Deep generative models have significantly advanced audio synthesis. Notable examples include AudioLDM (Liu et al.), a diffusion-based model, and MusicGen (Copet et al.), a transformer-based autoregressive model. MusicGen improves generation by conditioning on both textual and audio embeddings but lacks fine-grained control over adherence to conditioning, especially for audio-based guidance. Our work extends MusicGen (Copet et al.) by integrating composer-specific style embeddings for precise style transfer.

Symbolic Music and Style Transfer

Symbolic music, such as MIDI, has been widely used for style transfer. Zhu et al. introduced chord-aware transfer to preserve harmonic structure, while Brunner et al. used a VAE for disentangled MIDI representations. Kim et al. (2024) proposed music rearrangement methods, and Huang et al. explored diffusion-based multi-style transfer. However, these methods often suffer from slow audio generation and high compute requirements.

Spectrogram-Based Learning for Composer Classification

Spectrogram-based learning is effective for music classification. Choi et al. introduced a CNN-based approach to capture hierarchical composer-specific features. We build on this by integrating classification with generation, using a CNN-based style encoder.

Our novelty lies in: 1) Fine-tuning a pretrained model on a small dataset to reduce training cost and compute. 2)A unique style transfer approach that minimizes MSE between generated and target composer style embeddings using a custom encoder. 3) Objective evaluation via a classifier instead of subjective Mean Opinion Scores (as in (Huang et al.)), enabling faster, automated assessment.

3 Dataset and Features

Our study uses the MusicNet (Thickstun et al., 2017) dataset, a collection of 330 classical music recordings with over one million annotated labels. We use specific preprocessed versions of the dataset for the purpose of each task.

To facilitate the **composer classifier**, we preprocessed the dataset by extracting one-minute audio snippets starting after the first minute of recording. The extracted segments are processed using YAMNet (noa) for audio feature extraction. Our final dataset includes 1524 samples, with a 80/20 split for training and validation. For the **style encoder**, we utilized the first 60 seconds (= 960,000 features when sampled at 16K) of recordings and their corresponding composer labels. For **fine-tuning**, we constructed a dataset of 2880 paired samples by pairing each source composer with a target composer. We preprocess the first one minute of each song of the source composer and a generated text description (containing title + target composer name) using the Autoprocessor of MusicGen (Copet et al.) resulting in a tokenized input.

4 Methods

4.1 Composer Classifier

To evaluate whether the generated music successfully aligns with the target composer's style, we attempted multiple neural networks to classify music into the styles of 10 composers that are included in the dataset.

To train our composer classification model, we used **transfer learning** on the YAMNet model (noa). YAMNet (noa) is a sound classification model with 521 output classes such as "Whispering", "Frog", etc. unrelated to the classical music domain. We utilized YAMNet embeddings extracted from the dataset's audio recordings as input features.

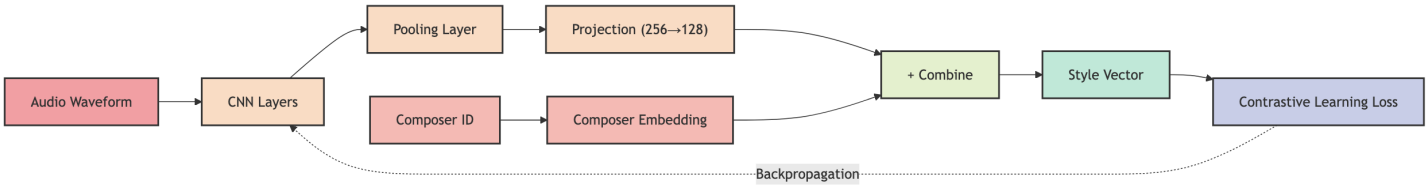

Figure 1: Architecture of style encoder

4.2 Style Encoder

The style encoder is designed to extract composer-specific stylistic features from input waveforms. It maps an audio waveform to a latent style representation to capture both the temporal structure and composer's identity.

The encoder processes raw waveforms through three 1D convolutional layers with ReLU activations. A final 1D convolutional layer enhances the temporal dependencies before we apply adaptive average pooling. The extracted feature vector is mapped into a latent space via linear projection layer, and the composer's identity is encoded using a learned embedding matrix of size (num_composers $\times$ output vector length):

$$z = W_p f(x) + E_c \tag{1}$$

, where $f(x)$ are the CNN-extracted features, W_p is the projection matrix, and E_c is the embedding matrix.

To enforce meaningful style separation in the latent space, we employ an Information Noise Contrastive Estimation (InfoNCE) loss function that optimizes the similarity between embeddings of the same composer and increases and distance between embeddings of different composer. For query q and a set of keys $\{k_0, k_1, k_2, ...\}$ where the match of q is k_+, the contrastive loss is defined as:

$$\mathcal{L} = -\log \frac{\exp(q \cdot k_+)/\tau}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)} \tag{2}$$

, where τ is a temperature hyperparameter.

4.3 Fine-tuning Music Generation

AudioLDM(Liu et al.) and MusicGen(Copet et al.) : Initially, our approach to music generation leveraged AudioLDM (Liu et al.), a latent diffusion model designed for text-to-audio generation. Our model consists of a CLIP-based text encoder that transforms texts into latent representation, an audio variational autoencoder that compresses the waveform, a fusion module that integrates textual and audio embeddings, and the UNet-based diffusion model.

However, the limit of AudioLDM (Liu et al.)exists in the fact that it does not support audio input, which is essential to our approach of style transfer. Therefore, we transitioned to MusicGen(Copet et al.) , a model that supports both text and audio conditioning.

MusicGen (Copet et al.) utilizes an autoregressive transformer architecture trained on EnCodec audio tokens:

$$p(z|c) = \prod_{t=1}^{T} p(z_t|z_{<t}, c) \tag{3}$$

, where c is the conditioning embeddings.

Finetuning with Style Encoder

We integrated the extracted composer embeddings from the style encoder into the MusicGen (Copet et al.) training process by incorporating them into the training objective. Specifically, we set the loss to be the Mean Squared Error (MSE) between:

1. The style embeddings generated by our Style Encoder on the audio produced by the model
2. The mean composer style embedding of the target composer (average of all style embeddings of audios corresponding to a given composer)

This approach guides the model to generate music that matches the stylistic characteristics of the target composer as captured by our Style Encoder. During fine-tuning, we optimize MusicGen(Copet et al.) to minimize the discrepancy between the generated and reference embedding. For efficiency, we employed Low-Rank Adaptation (LoRA (Hu et al., 2021)) with low-rank updates in attention layers:

$$\Delta W = AB, \quad A \in \mathbb{R}^{h \times r}, \quad B \in \mathbb{R}^{r \times h} \tag{4}$$

, where r is the rank of the update matrix.

5 Experiments

5.1 Composer Classifier

1. **Hyperparameter Tuning**: We experimented with different regularization values, learning rate exponential decay rates, different drop out rates, etc. The dataset was class-balanced using weighted cross-entropy, and we employed Adam optimizer with an exponentially decaying learning rate to prevent overfitting. We trained for 100 epochs with batch size equal to 64. After experimenting we used the following hyperparameters: The Adam optimizer uses an exponential decay schedule with an initial learning rate of 0.01, a decay rate of 0.96, and decay steps of 1000. For the MLP with L2 regularization we used $\lambda = 0.01$ and drop out equal to 0.3. For the residual blocks model, we used varying drop out rates and $\lambda = 0.001$ for certain layers.
2. **Model Architectures** The three models we trained using YAMNet embeddings are:
   - **Residual Block Model**: The model introduces skip connections, which allow gradients to flow more effectively and preserve information across layers.
   - **MLP with Regularization**: The model introduces batch normalization, L2 regularization, and increased dropout, aiming to improve generalization.
   - **Basic MLP Classifier**: The simplest model consists of a 3-layer feedforward network with fully connected layers.

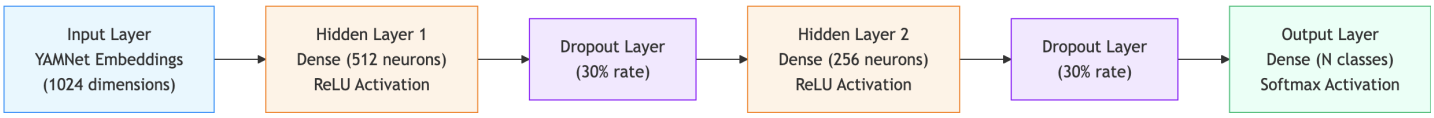

Figure 2: Structure of basic MLP composer classifier.

5.2 Style Encoder

1. **Architectures**: We experimented with different architectures that we chose due the following reasons

(a) **LSTM-based**: Effective for modeling long-term temporal dependencies, suitable for capturing stylistic variations over time.

(b) **GRU-based**: A computationally efficient alternative to LSTMs, retaining sequential modeling capabilities with fewer parameters.

(c) **CNN-based**: Extracts local patterns, leveraging hierarchical feature learning for style representation.

When experimenting with a subset of our dataset, we found that the CNN-based model was significantly faster to train. Therefore, we chose to train our entire dataset on the CNN-based model. We trained for 10 epochs. The dimension of the input audio to the classifier was [1, 9600000] and there were 10 composer classes. The outputted style embedding had dimension [1,128] resulting in significant dimensionality reduction. We used Adam optimizer with learning rate = 1e-4.

2. **Training a Composer Classifier using Style Embeddings**: In order to evaluate how well our style encoder's style embeddings represented the different composer styles, we also trained a simple MLP (same as 3-layer Basic MLP in Figure 1). We trained this classifier on the 2 one minute audio segments that followed the first minute of each audio recording in our dataset (discarding audios with length < 3 mins, no. of samples = 588, 80-20 train-test split).

5.3 Fine-tuning MusicGen(Copet et al.)

1. **Parameter Efficient Fine-Tuning (PEFT)**: Layers and Hyperparameters
   To increase the efficiency of fine-tuning, we leveraged Parameter-Efficient Fine-Tuning ((PEFT)), specifically, Low-Rank Adaptation (LoRA) method. We experimented with different target modules and lora alpha value in LoRA configuration. Initially, we targetted at a broader set of layers, including key(*k_proj*), value(*v_proj*), query(*q_proj*) projection, output(*out_proj*), feedforward layers(*fc1*, *fc2*), and the first layer of *lm_heads*. We also have a smaller scaling factor for low-rank updates with an alpha value of 16. Such configuration gives the model greater capacity to learn from the new data. However, due to the limit of computation power, we went for a minimal approach that focuses on the specific modules of key(*k_proj*) and value(*v_proj*) projection in the attention module. To complement the change, we increase our alpha value to 32, giving more impacts to each update. This experiment focuses on modifying the attention layer specifically, making the fine-tuning process lighter and more targeted at our goal of incorporating the style embeddings. We fine-tuned the model using AdamW with a high learning rate (0.1) and weight decay (1e-2).
2. **Dataset Sizing**: Latency Issues
   The original dataset includes 2880 samples with the waveform length of 1920000 (60 seconds sampled at 32K), which has a size of 30GB. With the limitation of 40.0 GB system RAM of A100 and computing units, we frequently ran into Out Of Memory problem during fine-tuning. We reduced the size to $1/4$ (7200) of the original dataset with random sampling. However, training was predicted to be > 7 hours. In the concern of disconnecting from the runtime, we further select 100 random samples for training.
   The fine-tuning process was conducted with Stochastic Gradient Descent (SGD). Since each forward pass (music generation) through the model was very slow due to the MusicGen(Copet et al.) architecture along with our compute limitations, we used a batch size of 1 and trained for one epoch.
3. **Evaluation Methods and Baselines** Fréchet Audio Distance (FAD) (Kilgour et al., 2019) and Composer Classifier.
   We measured the back-translation consistency of music through a circle of Composer A $\rightarrow$ Composer B $\rightarrow$ Composer A over 100 random samples. This approach determines whether the model truly captures Composer B's style and whether it can revert to Composer A with minimal distortion. FAD (Kilgour et al., 2019)then helps evaluate the quality of this process focusing on how close the reconstructed audio is to the inputted audio.
   Using the pretrained composer classifier, we get the percentage of generated pieces where the classifier's top prediction matches the target composer, and the percentage of cases where the classifier's top 2 predictions include the target composer. This evaluation is done on the 100 random samples as well.
   Our baseline model is the pre-trained MusicGen (Copet et al.) model without any fine-tuning.

6 Results and Discussion

6.1 Composer Classifier

Model	Accuracy. (%)	Precision. (%)	Recall (%)	F1-Score (%)
ResNet-Based	82.85	88.69	82.85	83.39
MLP w/ Regularization	**90.29**	**90.74**	**90.29**	**90.40**
Basic MLP Classifier	88.35	90.33	88.35	88.68

Table 1: Comparison of 3 Composer Classifiers

In table 1, We found that our the MLP with batch normalization, L2 regularization, and increased dropout performed better than the ResNet-Based model and the Basic MLP. We therefore used this composer classifier model for our evaluation of the finetuned MusicGen model.

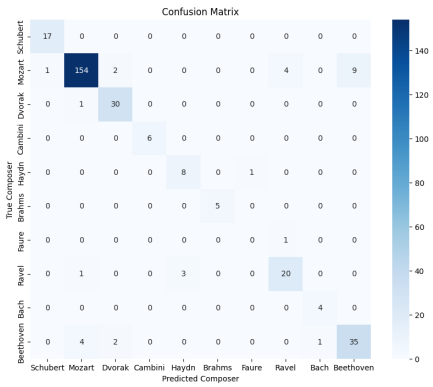

Figure 3: MLP w/ Regularization Confusion Matrix

6.2 Style Encoder

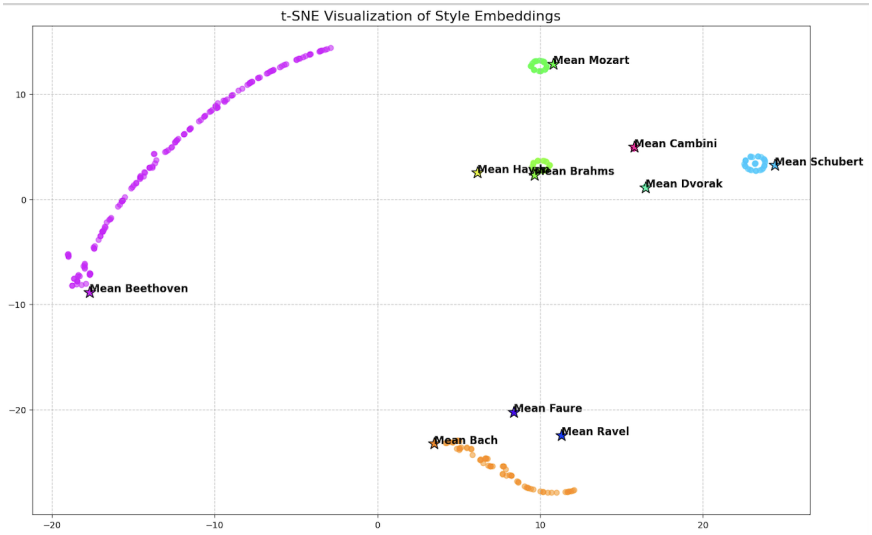

Figure 4: t-SNE visualization of style embeddings generated by Style Encoder of first 60 seconds of each recording in dataset

1. **CNN-based model**: In the t-SNE visualisation of the vector space, we see that embeddings of music by composers that are truly stylistically similar are close to each other (and different styles are further apart). For example, Faure and Ravel (Ravel studied composition under Faure!) have embeddings close to each other. Beethoven has a lot of stylistic variation across music, also reflected in figure. Thus, in a qualitative analysis we can see that the embeddings represent stylistic nuance well.
2. **Composer Classifier using Style Embeddings**: The MLP trained on the style embeddings achieved a training accuracy of 97% and validation accuracy of 95.54%. The weighted precision, recall, and F1 score were respectively 92%, 96% and 94%. Thus, we can see that our style embeddings represent stylistic nuance well.

6.3 Fine-tuning MusicGen(Copet et al.)

Model	Top-1 Acc. (%)	Top-2 Acc. (%)	FAD(Kilgour et al., 2019)(↓)
Baseline MusicGen	9.00	13.00	482.24
Fine-Tuned MusicGen (LoRA)	**16.00**	**23.00**	**329.13**

Table 2: Comparison of Fine-Tuned and Baseline MusicGen Models

The results in Table 2 highlight the improvement achieved through fine-tuning MusicGen(Copet et al.) on style transfer accuracy. The fine-tuned model achieves a 77.78% improvement in Top-1 accuracy and a 76.92% improvement in Top-2 accuracy compared to the baseline. For the FAD (Kilgour et al., 2019) score(the lower, the closer the generated audio is to the original), we observe a 23.14% decrease. This suggests that the fine-tuned model is better than the non-fine-tuned baseline at generating compositions that match the intended target composer.

7 Conclusion / Future Work

Our work demonstrates the effectiveness of finetuning to enhance MusicGen's ability to perform composer-specific style transfer by utilizing a novel training objective: minimizing Mean Square Error (MSE) loss between the style embedding of generated audio and the mean composer embedding, with embeddings generated by our custom style encoder. By leveraging a contrastive loss-based style encoder, we successfully extracted meaningful composer representations, allowing for more stylistically faithful music generation. Our composer classifier, trained using YAMNet embeddings, provided an objective evaluation metric, confirming improvements in stylistic adherence.

While our fine-tuning approach was constrained by computational limitations, our results suggest that increasing the dataset size and applying full fine-tuning could further enhance the model's capabilities. If we are offered more resources, we would fine-tune the model with the original dataset of 30GB, targeting more layers and would explore the possibility of fully fine-tuning the model.

8 Contributions

We both contributed equally to the project. We collaboratively worked on every single section of the project. We were both equally involved in the finetuning, data processing, ideation, model training, evaluation, and writing processes. All code can be accessed at

`https://github.com/SKAssist/CS229-project-final`

References

models/research/audioset/yamnet at master · tensorflow/models.

Gino Brunner, Andres Konrad, Yuyi Wang, and Roger Wattenhofer. MIDI-VAE: Modeling dynamics and instrumentation of music with applications to style transfer.

Keunwoo Choi, George Fazekas, Mark Sandler, and Kyunghyun Cho. Convolutional recurrent neural networks for music classification.

Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Défossez. Simple and controllable music generation.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Hong Huang, Yuyi Wang, Luyao Li, and Jun Lin. Music style transfer with diffusion model. Version: 1.

Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi. 2019. Fréchet audio distance: A metric for evaluating music enhancement algorithms.

Sooyoung Kim, Joonwoo Kwon, Heehwan Wang, Shinjae Yoo, Yuewei Lin, and Jiook Cha. 2024. A training-free approach for music style transfer with latent diffusion models.

Haohe Liu, Zehua Chen, Yi Yuan, Xinhao Mei, Xubo Liu, Danilo Mandic, Wenwu Wang, and Mark D. Plumbley. AudioLDM: Text-to-audio generation with latent diffusion models.

PEFT. PEFT.

John Thickstun, Zaid Harchaoui, and Sham M. Kakade. 2017. Learning features of music from scratch. In *International Conference on Learning Representations (ICLR)*.

Jinlong Zhu, Keigo Sakurai, Ren Togo, Takahiro Ogawa, and Miki Haseyama. MMT-BERT: Chord-aware symbolic music generation based on multitrack music transformer and MusicBERT.