Music Clip Interpolation with Variational Autoencoders

Krishna Penukonda *1001781*

Gao Yunyi *1002871*

Hong Pengfei *1002949*

Abstract

In this project, we aim to interpolate between 2 music excerpts by modelling Mel frequency spectrograms. Our main tasks are to use an autoencoder to get latent representations of music excerpts, interpolate between two latent vectors, decode the interpolated latent vector to Mel spectrogram and finally reconstruct audio from the resulting spectrogram. We employ a Variational Autoencoder to model the latent space in the hopes of being able to decode meaningful spectrograms from interpolated latent vectors.

Dataset and Data Collection

We use the open-source audio dataset called Free Music Archive (FMA) [1]. It is a library of high-quality, legal audio downloads directed by WFMU launched in 2009 [1]. It contains MP3 files of 77,643 songs covering 68 genres and sampled at 44,100Hz, 128kb/s. It contains not only the audio but also pre-calculated features for each clip (Table 1).

Figures 1 shows our graphical analysis of the genre distribution over 16 top genres in the dataset. Due to the unbalanced dataset with respect to genre, the team behind the FMA dataset has already defined a balanced subset over the 8 top genres in the metadata. We have filtered out the balanced subset and Figure 2 is our graphical visualization of the balanced subset which we use to train all of our models.

genre_top	number
International	10581
Blues	1818
Jazz	6256
Classical	6639
Old-Time / Historic	868
Country	3370
Pop	22824
Rock	80567
Easy Listening	1337
Soul-RnB	2640
Electronic	63220
Folk	21338
Spoken	4037
Hip-Hop	12036
Experimental	82674
Instrumental	28755

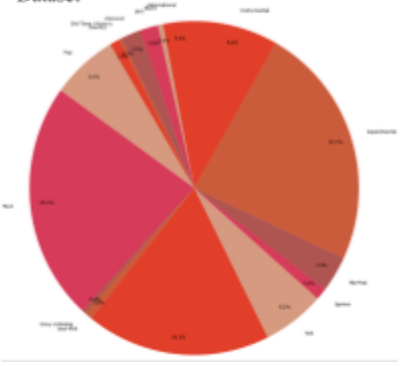


Figure 1: Genre Distribution in the FMA Dataset

genre_top	number
Electronic	5113
Experimental	7033
Folk	4504
Hip-Hop	7780
Instrumental	4306
International	4947
Pop	5582
Rock	4749

Figure 2: Balanced FMA Subset

Low-level Musical Features		High-level Musical Features	Other Metadata
Feature	**Dimensions**	Acousticness Danceability Energy Instrumentalness Liveness Speechiness Tempo Valence	Genre Album Artist
Chroma_Cens	83		
Chroma_CQT	83		
Chroma_STFT	83		
MFCC	139		
RMSE	6		
Spectral Bandwidth	6		
Spectral Centroid	6		
Spectral Contrast	48		
Spectral Rolloff	6		
Tonex	41		
ZCR	6		

***Table 1**: Music features included in the FMA Dataset*

Visualization of musical features

We visualize the balanced dataset distribution over 3 sets of features: *librosa features, temporal features and high-level audio features* (high level musical features including acousticness, danceability, energy, instrumentalness, liveness, speechiness, tempo, valence).

We use PCA and UMAP to perform dimensionality reduction for visualization. We ran UMAP on each feature set using a variety of hyperparameters and chose the hyperparameters that produced maximal separation of classes in the visualization.

Although we are able to distinguish genres based on these features using classification algorithms (as is described later in this report), it's quite hard to discriminate them from these visualizations of low- and high-level features.

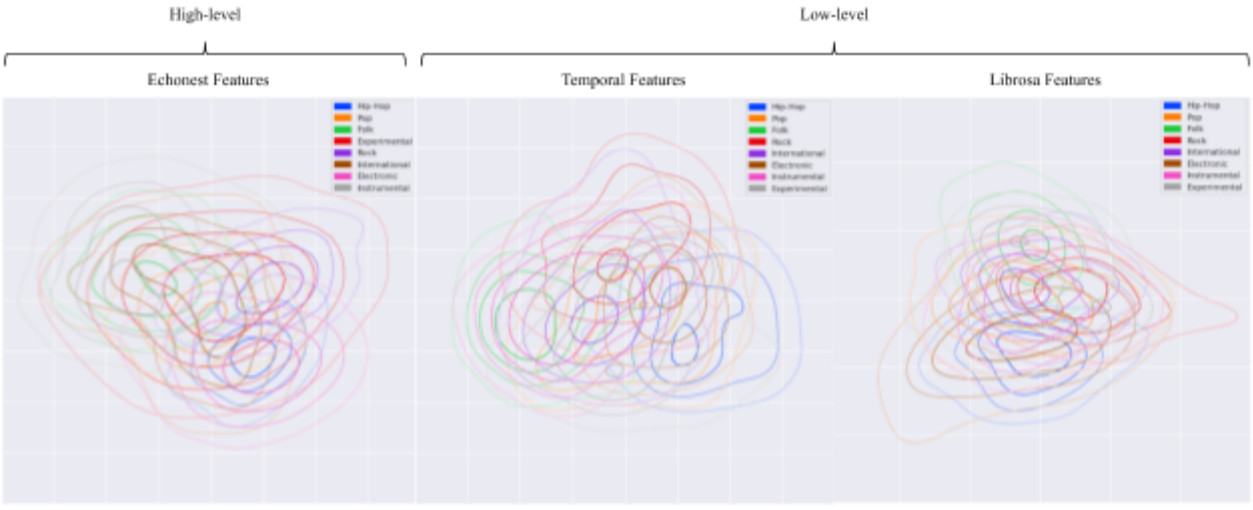


Figure 3: Visualization of musical features using PCA dimensionality reduction

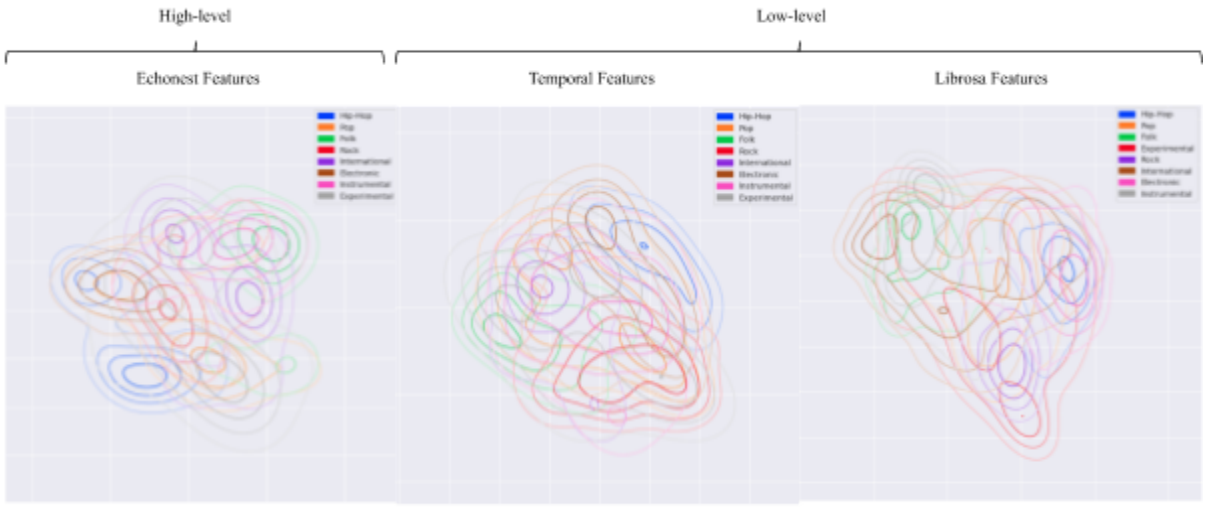


Figure 4: Visualization of musical features using UMAP dimensionality reduction

Data Pre-processing

In order to interpolate between audio samples, we need a generative model that's able to capture both the low-level and high-level features of audio clips. Earlier works such as WaveNet [4] aim to do this by directly modelling the individual samples in audio clips. Since audio in its raw form requires hundreds of thousands of samples to represent just a few seconds, and due to the inherently serial nature of audio, modelling it in such a manner means that the learning algorithm would need to learn the relationships between samples that are hundreds of thousands of time-steps apart.

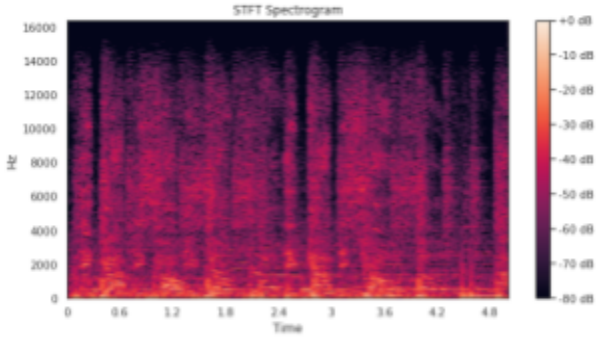


Figure 5: STFT Spectrogram

In order to enable our model(s) to capture higher-level information about audio clips, we convert the time-domain audio signal into a time-frequency domain spectrogram. We do this by sequentially applying an FFT over overlapping windows of the time-domain audio signal to obtain what's known as a Short-Time Fourier Transform (STFT).

To better represent the human perception of pitch, we rescale the frequency axis to use the Mel scale. [5] In addition, we take the logarithm of each value in the spectrogram to more closely match how humans perceive loudness and then normalize each spectrogram to the range [0, 1].

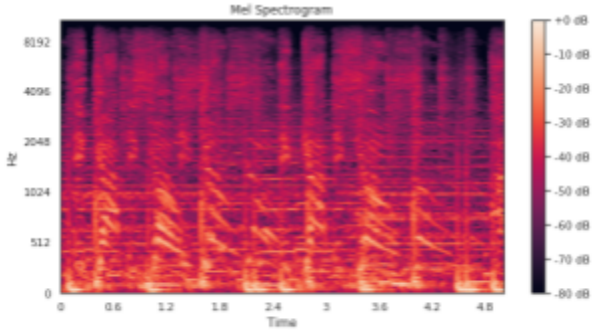


Figure 6: Mel Spectrogram

Since our goal is to be able to generate audio samples (and not just their spectrogram representations), we also need to be able to reconstruct audio clips from Mel spectrograms. Unfortunately, the reconstruction is unavoidably lossy because in deriving a Mel spectrogram we discard some temporal information

(FFT Phase) and some frequency information (in converting to the Mel scale).

To convert a Mel spectrogram to audio, we must first recover the removed frequency information by converting the Mel scale to a linear frequency scale. Thereafter, we reconstruct the phase information and apply the inverse FFT function to recover the audio. The Griffin-Lim algorithm is one method of estimating the FFT phase array based on the FFT magnitude array.

The Small subset of the FMA Dataset contains around ~8000 audio clips trimmed to approximately 30 seconds each. After removing ~10 clips that were malformed or unusable, we converted each one to its Mel spectrogram representation. We then split each spectrogram into subsections of 640 frames each, which with our FFT parameters corresponds to 5 seconds of audio. Splitting the spectrograms in this manner and discarding any subsections shorter than the specified 640 frames gives us a clean, consistent dataset. The genre-balanced nature of the small subset of the FMA Dataset we use means that we do not have to perform any rebalancing. As reconstructing a whole song will be much harder because of the long range dependencies, we tackle a simpler version of the problem by splitting the Mel spectrogram into 5 seconds and reconstructing them each separately.
We use `librosa` along with `numpy` to perform the majority of our data pre-processing. To ensure portability and to keep the disk-footprint of the generated spectrograms small, we store the generated spectrograms as OpenEXR images. OpenEXR is a lossless open image standard that supports 32-bit floating-point values, and is therefore ideal for holding spectrogram data.

When converting an audio clip to a Mel spectrogram, we note a tradeoff between reconstruction quality and reconstruction time. Using `librosa`, the spectrograms resulting from the parameters we chose take about 18 seconds to reconstruct (per second of audio) on a high-end consumer CPU.

The parameters we use for spectrogram generation are presented in Table 2.

Parameter	**Value**
Audio Sampling Rate	32768 Hz
FFT Window Size	256
STFT Hop Length	4096
Mel Frequency Bins	512

***Table 2**: Parameters for Mel spectrogram generation*

Problem and Algorithm / Model

Inspired by MusicVAE [3], a machine learning model that let us create palettes for blending and exploring musical scores using MIDI files, we intend to make a model that interpolate between two music excerpts using Mel spectrograms, which are more information-rich representations of audio.

Our tasks include audio reconstruction and audio interpolation. MusicVAE uses an architecture known as a Variational Autoencoder, which is capable of learning latent representations of a dataset. An autoencoder builds a latent space of a dataset by learning to compress (encode) each example into a latent vector, and then reproduce (decode) the same example from that latent vector. MusicVAE conducts reconstruction, sampling, and smooth interpolation based on this architecture.

Variational Autoencoder (VAE) for Latent Space Interpolation

In order to explore the variations of our dataset we use a variational autoencoder because a standard autoencoder generates encodings specifically useful for reconstructing its input - the loss function is usually either the mean-squared error or cross-entropy between the output and the input, known as the reconstruction loss, which penalizes the network for creating outputs different from the input.

In contrast, a VAE, rather than directly encoding a latent vector (say, of size n), encodes two vectors of size n each: a vector of means μ, and a vector of standard deviations σ. A latent vector is then randomly sampled from a multivariate gaussian distribution based on these vectors. In doing this, the decoder learns to map points in the neighborhood of a given spectrogram's latent vector to outputs that are similar to the given spectrogram. This allows the decoder to not just decode single, specific encodings in the latent space, but ones that slightly vary too.

The interpolation of two music excerpts (say A and B) is done by obtaining a linear combination of their spectrograms' latent vectors with some parameter *alpha* that controls the weight of each latent vector. The resulting latent vector is then decoded to produce an intermediate Mel spectrogram. The following section records our experiments on the above 2 tasks.

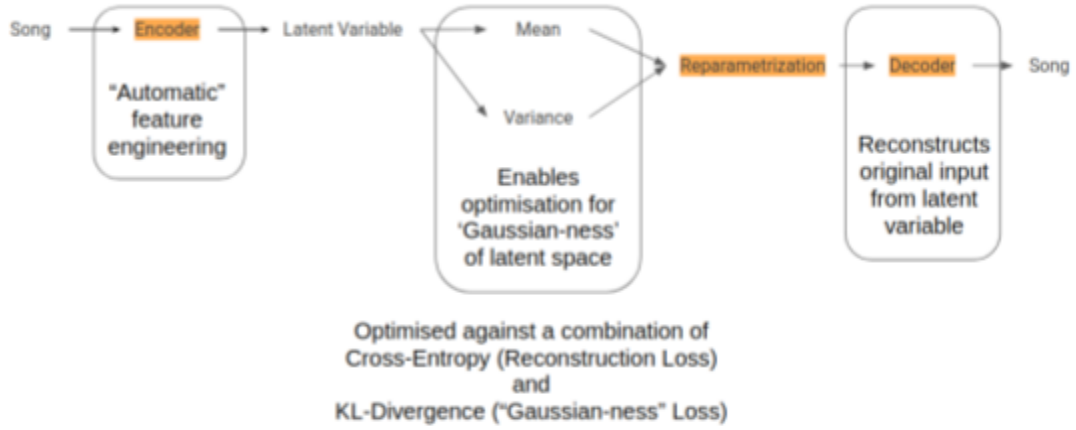


Figure 6: Variational Autoencoder

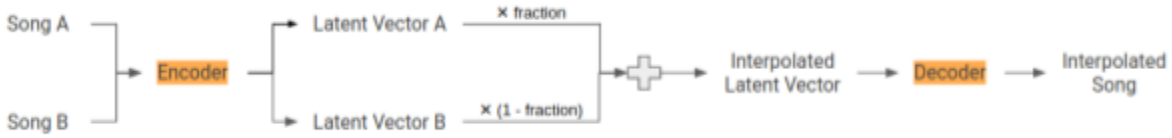


Figure 7: Latent space interpolation

Experiments

Baseline model

Our first attempt used a simple 3-layer 2D CNN as the encoder and 3 2D Transpose Convolutional layers as the decoder. This network when trained on a very small subset (~100 images) of the FMA dataset results yields reconstructions as shown below. We then attempted to interpolate between two audio samples via the latent space, but note that the reconstruction is too blurry to discern any features. After several experiments and attempts at hyperparameter tuning, we found that we had erroneously been using Binary Cross Entropy loss rather than RMSE loss. The former works well on binarized data, but is not suited for continuous data. After switching to RMSE loss, we were able to produce much clearer reconstructions (see the next section).

The baseline model uses a CNN. Audio spectrograms are inherently sequential, which means that patterns in the lower-left corner of a spectrogram do not mean the same thing as identical patterns in the top-right corner. CNNs are translation-invariant, whereas spectrograms are not. A better model would take into account the fact that patterns in different locations on the spectrogram have different meanings (pitch, time). Thus, in addition to improving the reconstruction using 2D convolutions, we also implement 1D convolution-based models to remove translation-invariance along the frequency dimension.

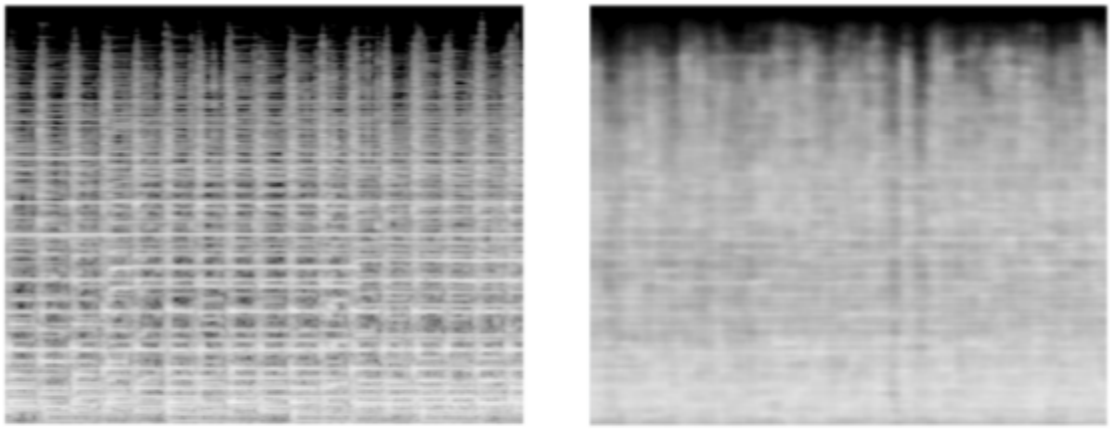


Figure 8: Baseline spectrogram reconstruction

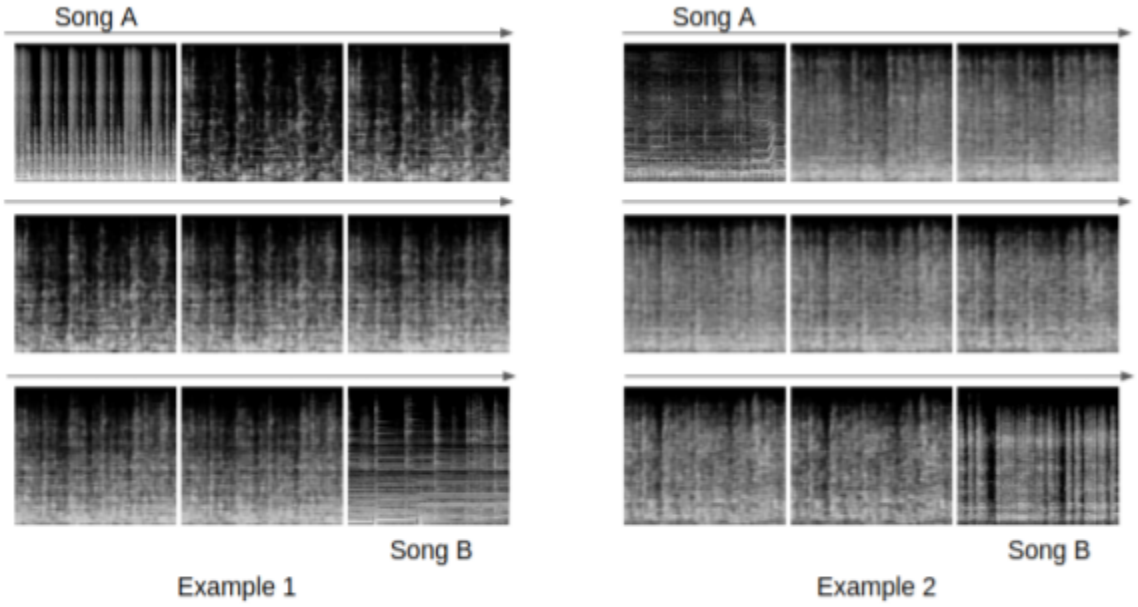


Figure 9: Baseline spectrogram interpolation.
Each set of 9 images is a different example

2D Convolution

Our second attempt is further exploration of CNN models using deeper neural networks. We use a 5-layer CNN. The input to our network is of size `[batch_size, frequency_dimension, time_dimension, 1]`.

Since we use strided convolutions, each subsequent layer of the encoder shrinks the image size (frequency and time dimensions), and the number of channels dimension increases to encode more features. At each layer of the CNN, we hope to encode higher level features about the image.

The reconstructed image is highly separated in every squared box. We find two possible reasons for this:

- We use transpose convolutions that have strides similar to their kernel size, so every step in that kernel will be similar.
- The current CNN layer have only one kernel size which is not able to capture all the features that can only be captured using other kinds of kernels.

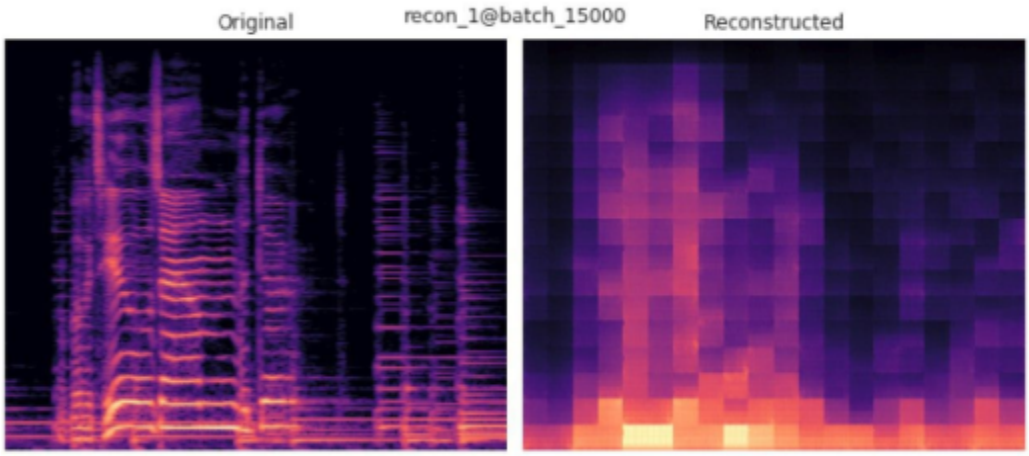


Figure 10: Reconstruction using naive 5-layer CNN with stride 2x2 at each layer

To address this issue of blocky reconstruction, we refer to the literature to find proven methods of implementing deeper convolutional neural networks for use in VAEs. Inception-VAE [8] is one such implementation, which uses Batch-Normalization. This implementation also organizes Conv2D layers into stacks (with control over how many layers are in each stack) with BatchNorm and ReLU activation at the end of each stack. The number of such stacks is fixed, with a Downsampling operation after each stack such that the network has a fixed downsampling ratio regardless of the number of convolutional layers.

Following this method, we obtain much clearer results, as shown in figures 11 through 13.

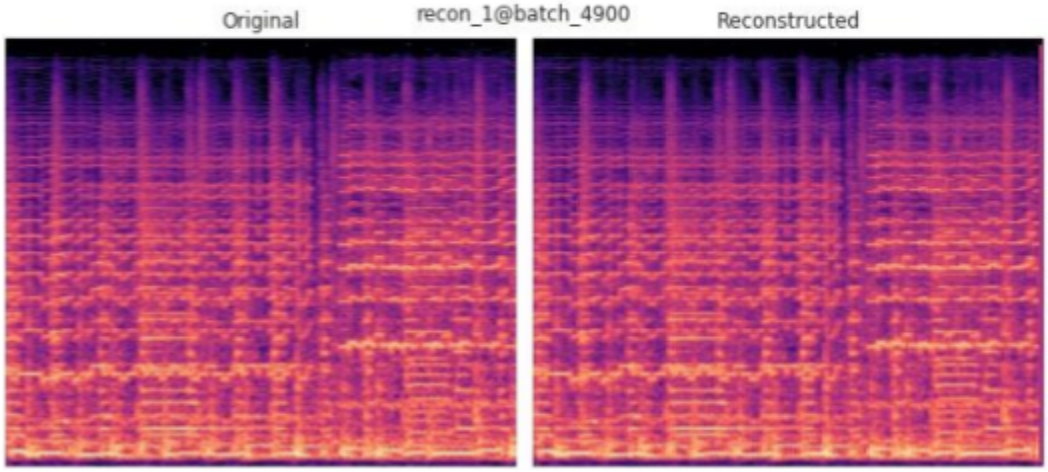


Figure 11: Conv2d reconstruction (trained on 1 sample)

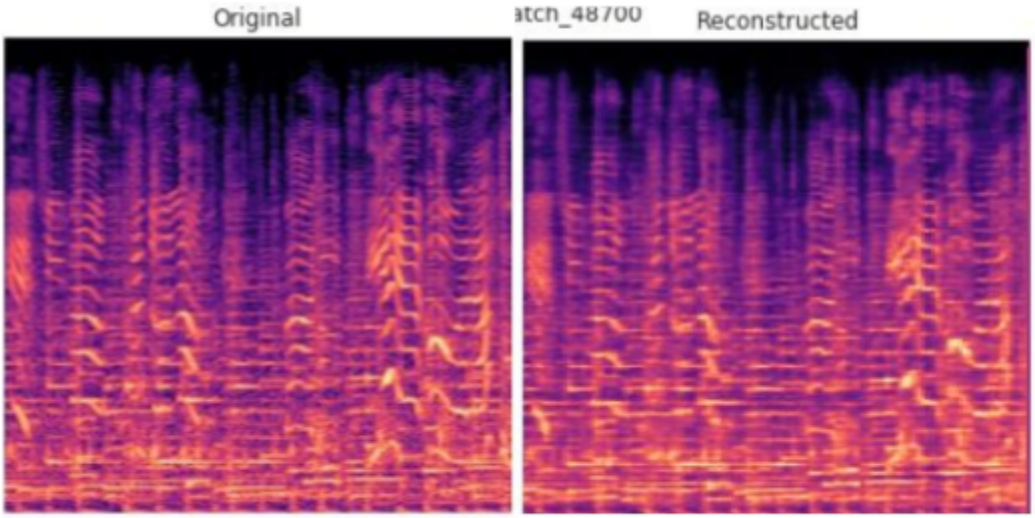


Figure 12: Conv2d reconstruction (trained on 20 samples)

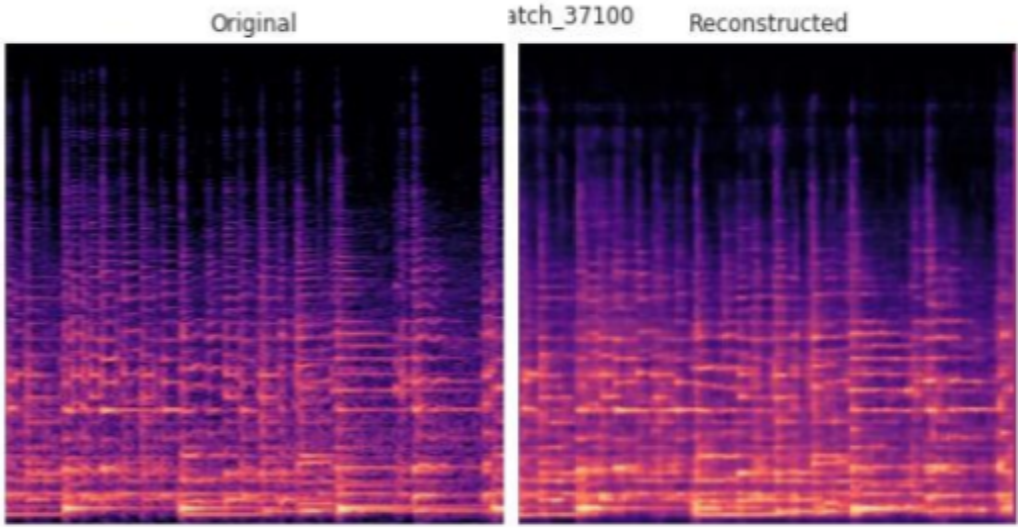


Figure 13: Conv2d reconstruction (trained on 100 samples)

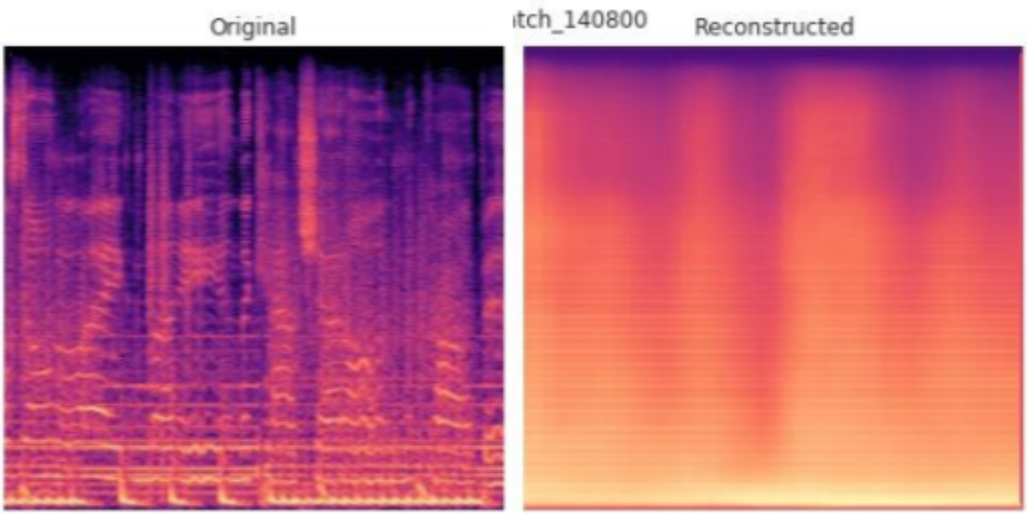


Figure 14: Conv2d reconstruction (trained on 39000 samples)

Conv2D + Inception

To increase the representational capacity of our model, we implement Inception modules, which consist of multiple parallel convolutional layers. Each of these layers has a different kernel size, and their outputs are merged to produce the Inception module's output. Unlike the original implementation[2], we sum the outputs of each parallel layer rather than concatenating them. This method preserves dimensionality and has been used with success by the team behind Inception-VAE. A diagram of the internals of the Conv2DBlock used in our model can be found in figures 14 and 15. The full graph of our network can be found in the appendix.

Whilst using the inception modules yields slightly sharper reconstruction, we also find that including these modules also makes the model more prone to generating artifacts in the output (figure 16). Due to time constraints, we were unable to further diagnose this issue and exclude the use of inception modules in all subsequent experiments.

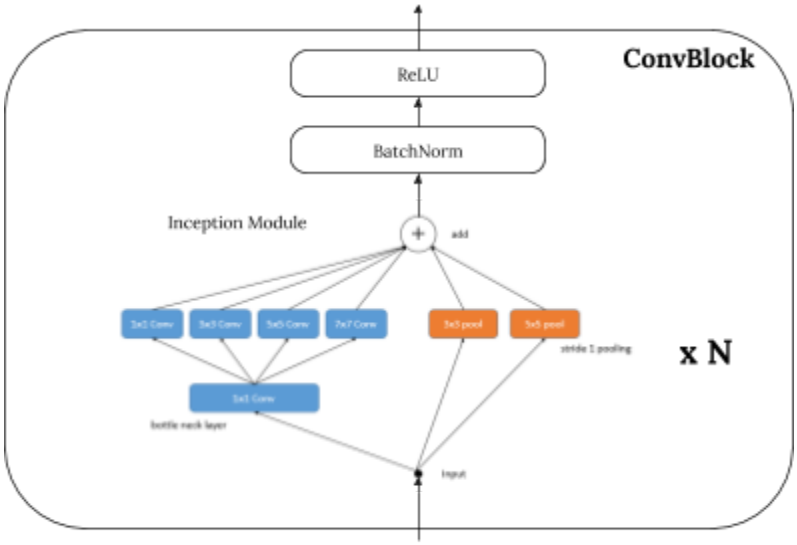


Figure 14: ConvBlock2D (with Inception modules)

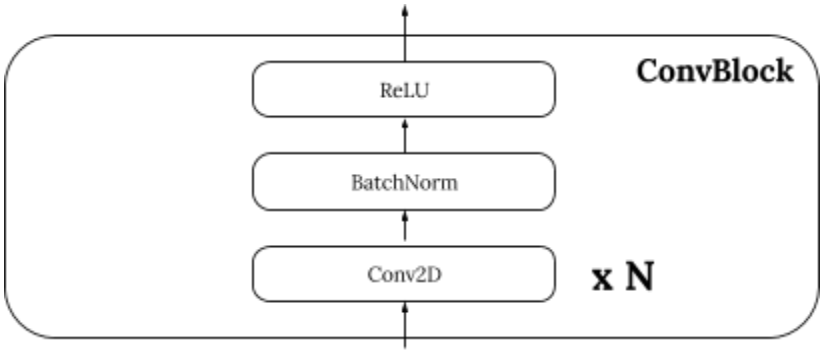


Figure 15: ConvBlock2D (without Inception)

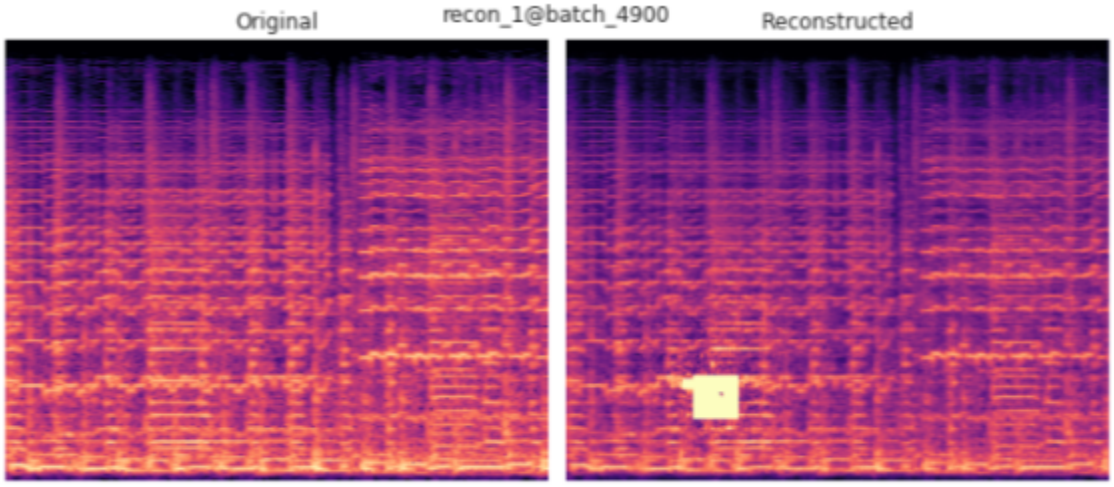


Figure 16: Reconstruction with Inception modules (trained on 1 sample). Note the artifact in the reconstructed spectrogram.

1D Convolution

Applying 1D convolution along the time-axis of the spectrograms should help eliminate the issue of translation invariance along the frequency axis. In order to ensure that all the frequency bins are modelled jointly by the convolution operations, we first transpose the input spectrogram from shape `[batch, freq, time, 1]` to `[batch, time, freq]`.
Unlike in the Conv2D case where the number of filters increases and size of the frequency and time dimensions decreases at each subsequent convolution block, we progressively decrease the number of channels so as to compress information about the frequency bins. The output we get from our Conv1D model shows very little differentiation along the frequency axis, but models the temporal axis fairly accurately.

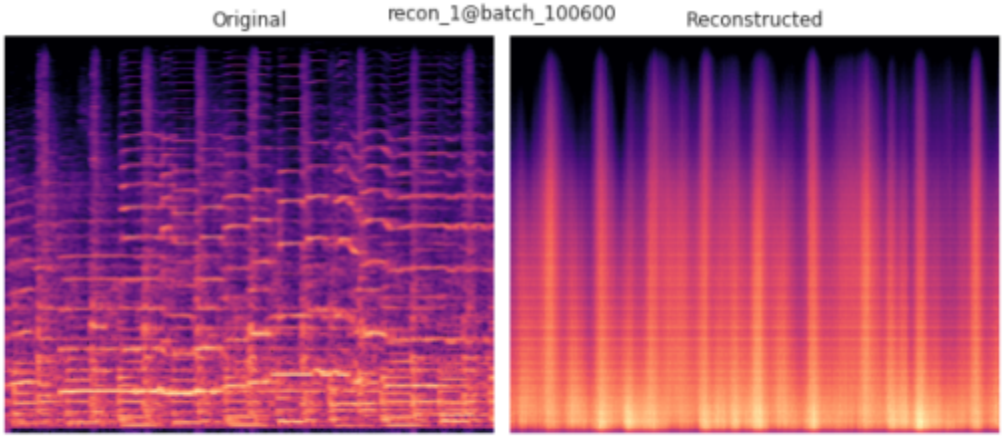


Figure 17: Reconstruction using 1D convolution (trained on the full dataset)

Conv1D + Inception

In order to improve sharpness and increase the representational capacity of our Conv1D model, we refactored the Inception-VAE network to use 1D Convolutional layers. Unfortunately, we noted no significant difference in reconstruction quality with the addition of inception modules, although model training time increased by a factor of ~4.

LSTM

As a final attempt at improving reconstruction sharpness, we replace the convolution-based encoder and decoder with LSTM-based ones. The full network graph can be found in the appendix. We treat the time-axis of our spectrograms as such when we feed them into the LSTM layers, i.e. each frame of a spectrogram is a single time-step from the LSTM layer's perspective. We find that the reconstructions produced by this approach have decent vertical differentiation, but temporal features appear to be smeared out.

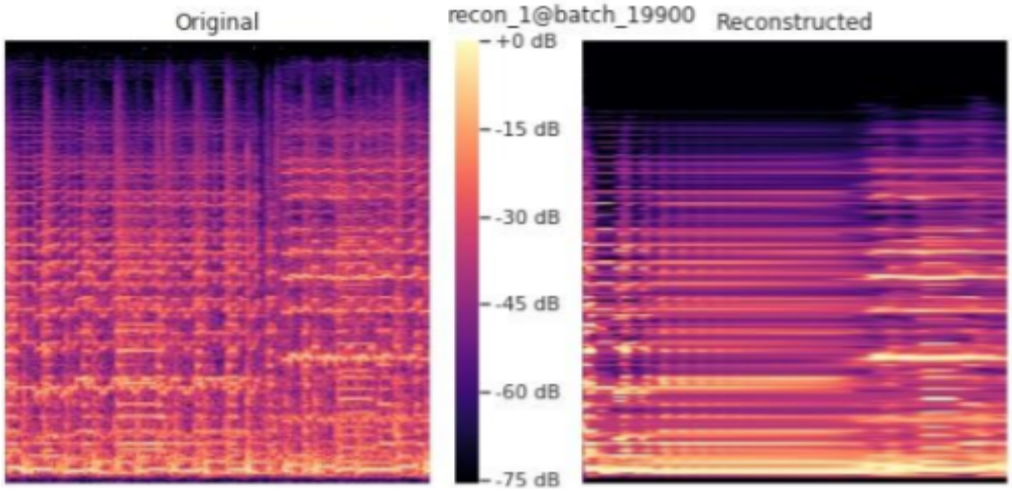


Figure 18: Reconstruction using LSTM-VAE

Hyperparameter Tuning

We tune each of our models by trying varying numbers of convolutional layers in both the encoder and decoder, as well as by varying the size of the latent vector and batch size. We tested with latent vector sizes [8, 256, 4096], and found that in general, larger latent vectors allow for marginally sharper reconstruction, but were not the limiting factor.

We also tried using ReLU activation as opposed to Sigmoid to test whether the Sigmoid activation may be forcing the values in the output to the extremes, rather than producing a smooth distribution of intensities. We find that both activation functions produce similar results after a few epochs of training, and the distribution of intensities in the output resembles the input distribution to roughly the same degree in both cases.

In addition, we experimented with the different padding schemes offered by Tensorflow for convolutional layers ("SAME" and "VALID"), as well as two different kinds of pooling layers at the end of the encoder (Max Pooling and Average Pooling).

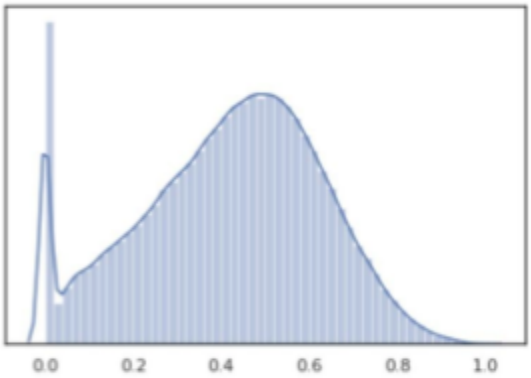
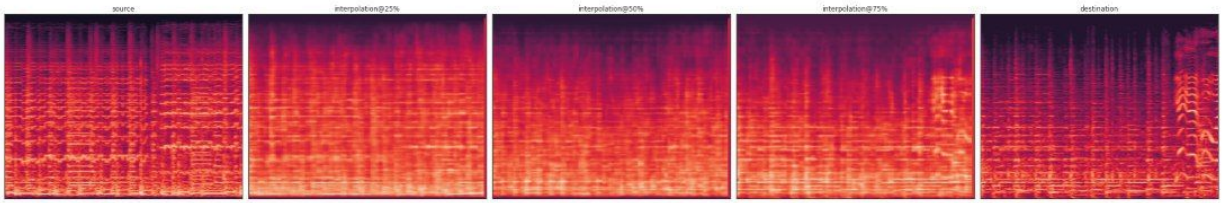


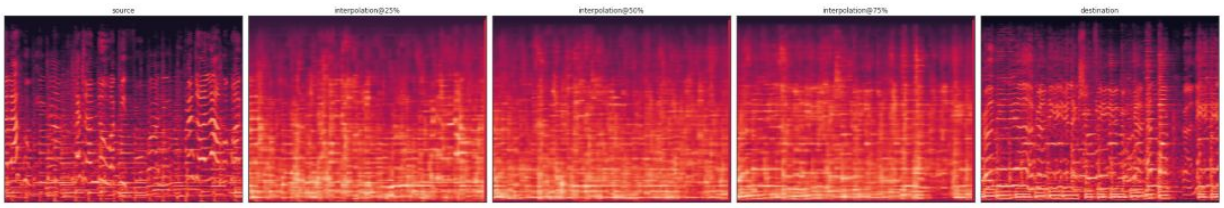
Figure 17: Reconstructed spectrogram intensity distribution.

Music Interpolation Results

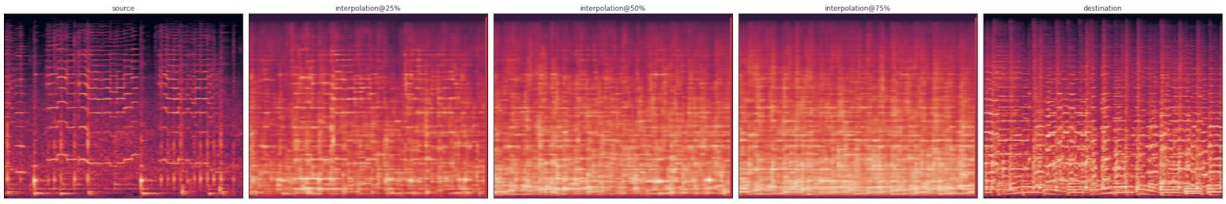
Using our final VAE model *(see Appendix)*, we extract the latent vector representations of two music clips' spectrograms. We then perform a linear combination of these latent vectors with some parameter alpha that controls the weight of each latent vector to generate an interpolated latent vector and decode it to a Mel spectrogram. Below are examples of interpolation using two different music excerpts. The first and last spectrogram are from the original music excerpts (A and B respectively). The middle three mel spectrograms are generated ones, transitioning from left to right.
Audio samples reconstructed from below examples can be listened to at
https://github.com/tasercake/BeatBrain/blob/master/results/results.md



Example 1 (https://github.com/tasercake/BeatBrain/tree/master/results/1)



Example 2 (https://github.com/tasercake/BeatBrain/tree/master/results/2)



Example 3 (https://github.com/tasercake/BeatBrain/tree/master/results/3)

Evaluation

Methodology

Generative models are inherently difficult to evaluate quantitatively. We apportion the evaluation into 3 sections, each of which corresponds to a different modelling/generation task:

1) Evaluating the reconstruction of spectrograms (and by extension, audio) via our VAE model
2) Evaluating the unconditioned generation of spectrograms and audio
3) Evaluating the results of latent space interpolation between audio clips

Image Similarity

To quantify the spectrogram reconstruction quality, we use three measures of image similarity:

Training Samples	**Normalized Cross-Correlation**	**Peak Signal-to-Noise Ratio**	**Structural Similarity Index**
1	0.9421	-42.383	0.8522
20	0.7960	-48.897	0.5450
100	0.6427	-53.785	0.3517

We note from the above results that the reconstruction quality drops significantly as we increase the number of training samples. This is a hurdle we have not yet overcome and are therefore unable to produce useful results over the full dataset.

Genre Classification

Music genre is an important high-level perceptual characteristic of music. Different music genres are often associated with different high-level musical attributes and human perception of the music. Our assumption is that a good autoencoder model is able to capture important high level aspects of the spectrogram, and that we should be able to use a simple machine learning model to classify genre using the latent vectors output by the *encoder* portion of the autoencoder. The expectation is that classification performance using this method would be superior to that obtained using traditionally extracted musical features.

Furthermore, assuming that our classifier outputs a vector of probabilities for each genre, we can posit that running the classifier over a spectrogram C derived via latent space interpolation between two other spectrograms A and B would result in a genre probability vector that lies in between the genre probability vectors of A and B.

Therefore, if a machine learning method is able to do genre classification directly from the high- or low-level features (such as the Echonest features or Librosa features), and we can also use the same algorithm to predict genre directly from the latent vectors, it can be an important indicator that our model has encoded those features in the latent vectors. Table 3 shows the performance of various classification algorithms on musical features.

To get a clearer picture of the VAE's representational capacity, we would train each of these classifiers using the VAE's latent vectors as inputs, but due to time constraints and the fact that our VAE model doesn't produce usable reconstructions when trained on the full FMA dataset, we leave this task for the future.

Algorithm	**Class-Weighted F1 Score**
Traditional Machine Learning Different parameters are tried for each method, with only the best result shown. The mean, standard deviation, and median of each frequency band in the Mel spectrogram were used as input features.	
Decision Tree	0.26
Logistic Regression	0.37
SVM	0.59
KNN	0.35
Gaussian Naive Bayes	0.24
Gaussian Mixture Model	0.15
Deep Learning Trained on 35000 samples (raw Mel spectrograms), validated on 8320 samples. Dataset is pre-balanced, and features are normalized over the entire dataset.	
CNN_1	0.41
CNN_2	0.46
RNN	0.41

***Table 3**: Genre classification accuracy using various machine learning techniques*

Given above attempts, SVM (with regularization parameter C equals to 100) gives the best validation accuracy. To tackle the issue of overfitting, we apply the following techniques:

1. Adding dropout layer
2. Adding regularization to the Dense layer
3. Early stopping

Methods 1 & 3 work quite well and prevent overfitting. We chose the best hyperparameters via testing on the validation set, and measure the models' performance on the test set.

It seems that deep learning models are easily ovefit to the training data, and using machine learning methods on high level audio features seems to solve this problem.

User Surveys

The spectrograms produced by the latest iteration of our VAE model are rather obviously poor in quality, and the audio reconstructed from them is similarly noisy. If and when the model is improved, it would be helpful to be able to quantify the quality of audio reconstruction, generation, and interpolation.

To do this, we propose a listening survey where we ask users to rate music samples based on the following cues:

1. Does the generated music sound clean?

2. Do the interpolated music samples transition fluently?
3. Does this music make you feel uncomfortable?
4. Do the original and reconstructed audio have similar qualities such as tempo, melody, beat, and loudness?

Results and Discussion

We have tried with KL divergence loss and without KL divergence loss, the reconstruction works better without KL divergence loss, and the model can construct clearer images. The reason is because KL divergence will try to make the latent variable closer to the Guassian distribution to make the decoder generate meaningful results from any vectors from random normal distribution. So we will still keep the KL-divergence loss. But we think the music is a complex data, it is hard to map it to a simple normal Gaussian distribution. We think it may help if we can use variational autoencoder with Gaussian Mixture Model. In other words, each song is mapped to a different Gaussian distribution.

In experiments with latent vector generator, we have attempted with implementation of LSTM as encoder and decoder for sequential data, with inception layer, with Conv1D and Conv2D. We have evaluate the similarity in extracted features / genres between original audios and reconstructed audios. Our models work perfectly when overfit over small sample size like 200, but shows blurry decoded mel-spectrogram and reconstructed audio containing fewer characteristics over the entire dataset, which might be caused by large dataset of abundant music genres for autoencoder. In future experiments, smaller dataset of more singular type should be tested for generative model. In addition, in future development, we could also try to run post-processing on generated audio output to remove noise.

Future Directions

We are far from ideal to do produce sensible music that can be appreciated by humans, one of the reasons is that the FMA dataset is too complicated, one audio may contain lyrics as well. It will be useful if we can seperate the lyrics from the music and focus on producing the music itself. One potential dataset we can experiment on is the **Groove dataset** [6] with only instrumental sounds.

As the VAE models "explicit" distribution of the data by trying to fit the data via a multi-dimensional Gaussian/Normal distribution. However, Generative Adversarial Networks learn an "implicit" distribution of data meaning that you cannot directly sample them. So we can progress to do model training on **VAE-GANs** which can provide clearer image and results. [7] we also can model the frequency domain just like melnet.

Variational autoencoders + GMM

Each Gaussian in the Gaussian mixture corresponds to a different cluster. If each music genre is sampled from a different Gaussian Distribution, Gaussian mixture will have a latent space (z), that is more fit to model the distribution of the multi-genre dataset, and therefore produce better results.

References

[1] Defferrard, M., Benzi, K., Vandergheynst, P., & Bresson, X. (2016). Fma: A dataset for music analysis. *arXiv preprint arXiv:1612.01840*.

[2] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

[3] Roberts, A., Engel, J., Raffel, C., Simon, I., & Hawthorne, C. (2018). MusicVAE: Creating a palette for musical scores with machine learning, March 2018. (https://magenta.tensorflow.org/music-vae)

[4] Bakshi, B. R., & Stephanopoulos, G. (1993). Wave-net: a multiresolution, hierarchical neural network with localized learning. *AIChE Journal*, *39*(1), 57-81. (https://deepmind.com/blog/article/wavenet-generative-model-raw-audio)

[5] McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M., Battenberg, E., & Nieto, O. (2015, July). librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference* (Vol. 8).

[6] https://magenta.tensorflow.org/datasets/groove

[7] https://towardsdatascience.com/what-the-heck-are-vae-gans-17b86023588a

[8]https://github.com/koshian2/inception-vae

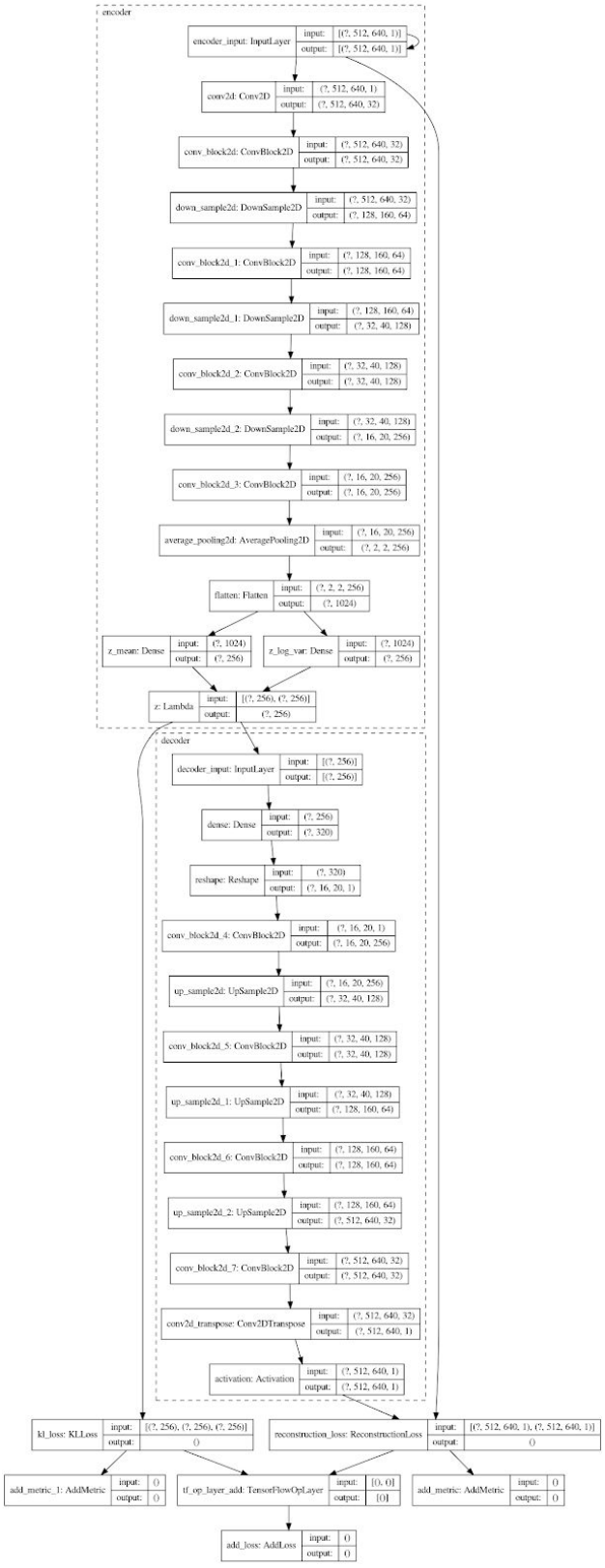
Appendix

A. Model Architectures

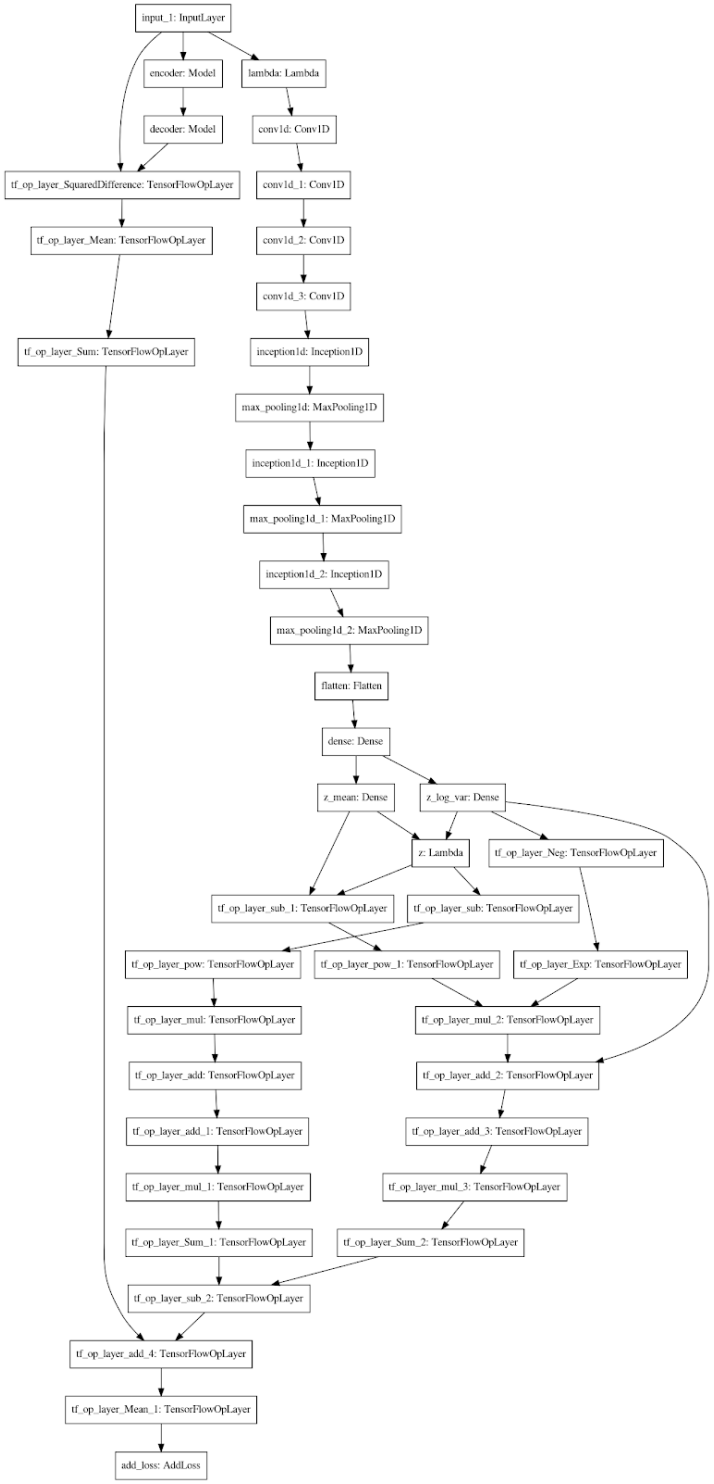
Baseline Model Architecture

```
Model: "Encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 255, 319, 32)      320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 127, 159, 64)      18496
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 63, 79, 64)        36928
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 31, 39, 64)        36928
_________________________________________________________________
flatten (Flatten)            (None, 77376)             0
_________________________________________________________________
dense (Dense)                (None, 64)                4952128
=================================================================
Total params: 5,044,800
Trainable params: 5,044,800
Non-trainable params: 0
_________________________________________________________________
Model: "Decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 81920)             2703360
_________________________________________________________________
reshape (Reshape)            (None, 32, 40, 64)        0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 64, 80, 64)        36928
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 128, 160, 64)      36928
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 256, 320, 64)      36928
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 512, 640, 32)      18464
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 512, 640, 1)       289
=================================================================
Total params: 2,832,897
Trainable params: 2,832,897
Non-trainable params: 0
_________________________________________________________________
```

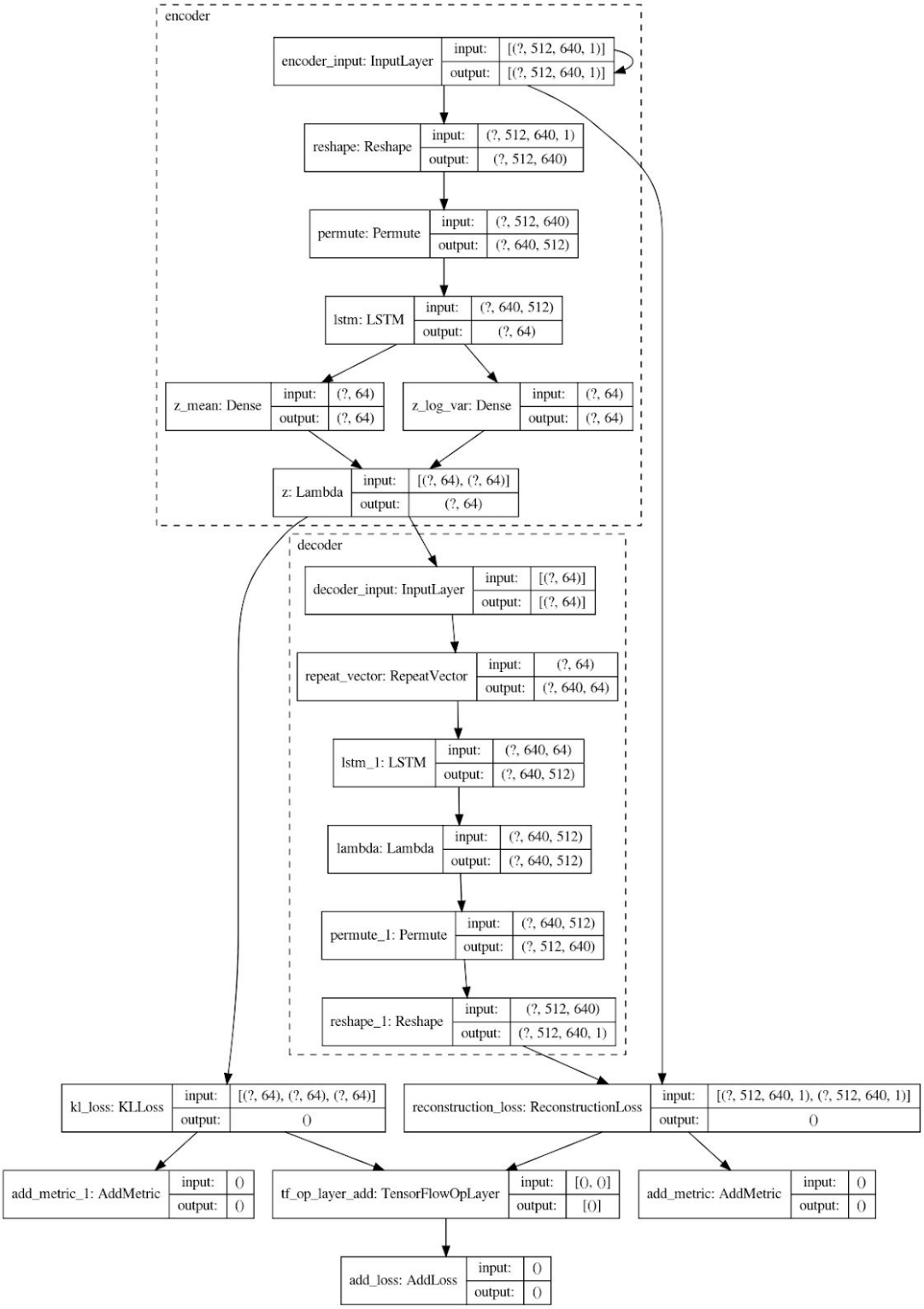
Final Conv2D-VAE Architecture



Conv1D-VAE Architecture



LSTM-VAE Architecture



B. Genre Classification Experiment Record

Traditional Machine Learning

Algorithm	**Hyperparameters**	**Results**
Decision Tree	`leaf_ls=[1,3,5,7,10,25]`	min leaves=10 precision recall f1-score support 0 0.31 0.25 0.28 297 1 0.33 0.29 0.31 401 2 0.20 0.36 0.26 146 3 0.23 0.18 0.20 277 4 0.17 0.36 0.23 134 5 0.32 0.19 0.24 312 6 0.17 0.14 0.15 304 7 0.34 0.50 0.41 129 accuracy 0.26 2000 macro avg 0.26 0.28 0.26 2000 weighted avg 0.27 0.26 0.25 2000
Logistic Regression	`C_param_range = [0.001,0.01,0.1,1,10,100]`	Regularization parameter(C)=0.1 precision recall f1-score support 0 0.41 0.34 0.37 297 1 0.32 0.17 0.22 401 2 0.37 0.47 0.41 146 3 0.38 0.42 0.40 277 4 0.23 0.60 0.33 134 5 0.55 0.40 0.47 312 6 0.22 0.20 0.21 304 7 0.43 0.64 0.51 129 accuracy 0.35 2000 macro avg 0.36 0.40 0.37 2000 weighted avg 0.37 0.35 0.35 2000
SVM	`C_param_range = [0.001,0.01,0.1,1,10,100,150.200]`	Regularization parameter(C)=100 precision recall f1-score support 0 0.53 0.60 0.56 328 1 0.54 0.44 0.48 380 2 0.67 0.67 0.67 431 3 0.63 0.66 0.65 316 4 0.61 0.65 0.63 456 5 0.72 0.58 0.64 340 6 0.48 0.52 0.50 426 7 0.61 0.62 0.62 323 accuracy 0.59 3000 macro avg 0.60 0.59 0.59 3000 weighted avg 0.60 0.59 0.59 3000
KNN	`n_neighbours= [5, 10,20,40]`	n_neighbors=40 precision recall f1-score support 0 0.40 0.26 0.32 297 1 0.25 0.13 0.18 401 2 0.39 0.49 0.43 146 3 0.31 0.49 0.38 277 4 0.22 0.46 0.30 134 5 0.51 0.46 0.48 312 6 0.23 0.15 0.18 304 7 0.40 0.64 0.49 129 accuracy 0.34 2000 macro avg 0.34 0.39 0.35 2000 weighted avg 0.34 0.34 0.32 2000
Guassian Naive Bayes		precision recall f1-score support 0 0.24 0.05 0.09 297 1 0.36 0.04 0.07 401 2 0.15 0.45 0.23 146 3 0.31 0.50 0.39 277 4 0.28 0.67 0.39 134 5 0.31 0.06 0.10 312 6 0.19 0.10 0.13 304 7 0.24 0.91 0.38 129 accuracy 0.24 2000 macro avg 0.26 0.35 0.22 2000 weighted avg 0.27 0.24 0.18 2000

GMM		

	precision	recall	f1-score	support
0	0.00	0.00	0.00	297
1	0.25	0.02	0.04	401
2	0.00	0.00	0.00	146
3	0.00	0.00	0.00	277
4	0.00	0.00	0.00	134
5	0.00	0.00	0.00	312
6	0.15	0.99	0.27	304
7	0.00	0.00	0.00	129
accuracy			0.15	2000
macro avg	0.05	0.13	0.04	2000
weighted avg	0.07	0.15	0.05	2000

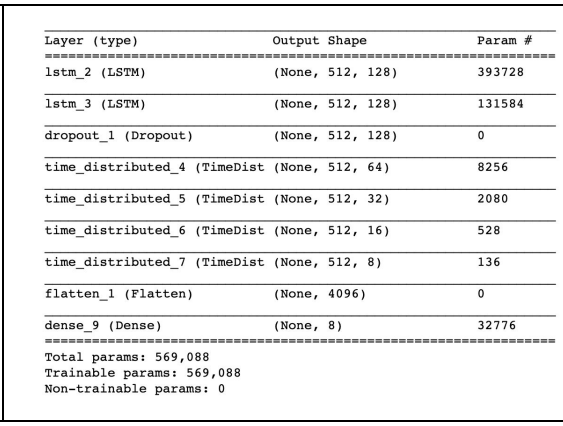
GMM + K-means Init		

	precision	recall	f1-score	support
0	0.15	1.00	0.26	297
1	0.00	0.00	0.00	401
2	0.00	0.00	0.00	146
3	0.00	0.00	0.00	277
4	0.00	0.00	0.00	134
5	0.00	0.00	0.00	312
6	0.00	0.00	0.00	304
7	0.00	0.00	0.00	129
accuracy			0.15	2000
macro avg	0.02	0.12	0.03	2000
weighted avg	0.02	0.15	0.04	2000

Deep Learning

Train on 35000 samples, validate on 8320 sample over 43000 samples in balanced dataset, features are normalized over the entire dataset.

Architecture		Results		Remarks
		Train Acc.	**Val Acc.**	
CNN_1	Layer (type) / Output Shape / Param # conv2d_18 (Conv2D) multiple 160 conv2d_19 (Conv2D) multiple 2320 conv2d_20 (Conv2D) multiple 2320 max_pooling2d_6 (MaxPooling2 multiple 0 dropout_6 (Dropout) multiple 0 flatten_6 (Flatten) multiple 0 dense_18 (Dense) multiple 167772288 dense_19 (Dense) multiple 8256 dense_20 (Dense) multiple 520 Total params: 167,785,864 Trainable params: 167,785,864 Non-trainable params: 0	0.6301	0.4128	Overfitting after ~3 epochs
CNN_2	Layer (type) / Output Shape / Param # conv2d_38 (Conv2D) multiple 303 conv2d_39 (Conv2D) multiple 4515 conv2d_40 (Conv2D) multiple 97565 flatten_12 (Flatten) multiple 0 dense_28 (Dense) multiple 2017608 Total params: 2,119,991 Trainable params: 2,119,991 Non-trainable params: 0	0.5969	**0.4561**	Overfitting after ~5 epochs. Conv2D may not be ideal for time-series features - RNNs may be more appropriate.

RNN		0.5230	0.4106	Overfitting after ~5 epochs